NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

SHEPHERD ROTARY VEHICLE: MULTIVARIATE MOTION CONTROL AND PLANNING

by

Edward J. Mays
and
Ferdinand A. Reid

September 1997

Thesis Advisor: Yutaka Kanayama

Approved for public release; distribution is unlimited.

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE September 1997	3. REPORT TYPE AND DATES COVERED Master's Thesis

4. TITLE AND SUBTITLE
SHEPHERD ROTARY VEHICLE: MULTIVARIATE MOTION CONTROL AND PLANNING

5. FUNDING NUMBERS

6. AUTHOR(S)
Mays, Edward J., and Reid, Ferdinand, A.

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Millions of acres of the US formerly used defense sites (FUDS) are contaminated with unexploded ordnance (UXO) as a result of past military use. The process of returning the land to the civilian sector is sensitive, intensive, and costly (e.g., millions of dollars, and the loss of human life). Hence "clearing" (i.e., site remediation, range clearance, and explosive ordnance disposal) UXO's from FUDS is a complex problem. Existing clearing methods are inaccurate, dangerous, and labor intensive. This thesis shows that through robotics technology (e.g., "Shepherd" rotary vehicle with three degrees of freedom) and the use of advanced computer technology it is possible to make clearing tasks safer, more cost-effective, and more efficient. An over arching hardware and software architecture was developed for Shepherd (including a self-contained on-board computer system). The software system was developed for timer control, motion control, user interface, and an operating kernel. The hardware and software organization, structure, and interaction provide the framework for real-time control. This research included the use of encoders, digital boards, and a counter board; required the handling of interrupts, electric motor manipulation by servomotor controllers, and communication using RS232 and VMEbus technology. The kinematics algorithms and a real-time operating kernel were implemented using the C language. "Shepherd" research has laid the foundation for the flexible, robust, and precise motion needed for UXO clearing.

14. SUBJECT TERMS
Unexploded Ordnance, Artificial Neural Networks

15. NUMBER OF PAGES
305

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18

Approved for public release; distribution is unlimited

SHEPHERD ROTARY VEHICLE: MULTIVARIATE MOTION CONTROL AND PLANNING

Edward J. Mays
Major, United States Marine Corps
B.S., University of Florida, 1984
M.S., University of Southern California, 1994

Ferdinand A. Reid
Lieutenant, United States Navy
B.B.A., Georgia State University, 1990

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
September 1997

Authors: ______________________
Edward J. Mays

____
Ferdinand A. Reid

Approved by: ______________________
Yutaka Kanayama, Thesis Advisor

____
Xiaoping Yun, Second Reader

____
Ted Lewis, Chairman
Department of Computer Science

ABSTRACT

Millions of acres of the US formerly used defense sites (FUDS) are contaminated with unexploded ordnance (UXO) as a result of past military use. The process of returning the land to the civilian sector is sensitive, intensive, and costly (e.g., millions of dollars, and the loss of human life). Hence "clearing" (i.e., site remediation, range clearance, and explosive ordnance disposal) UXO's from FUDS is a complex problem. Existing clearing methods are inaccurate, dangerous, and labor intensive. This thesis shows that through robotics technology (e.g., "Shepherd" rotary vehicle with three degrees of freedom) and the use of advanced computer technology it is possible to make clearing tasks safer, more cost-effective, and more efficient. An over arching hardware and software architecture was developed for Shepherd (including a self-contained on-board computer system). The software system was developed for timer control, motion control, user interface, and an operating kernel. The hardware and software organization, structure, and interaction provide the framework for real-time control. This research included the use of encoders, digital boards, and a counter board; required the handling of interrupts, electric motor manipulation by servomotor controllers, and communication using RS232 and VMEbus technology. The kinematics algorithms and a real-time operating kernel were implemented using the C language. "Shepherd" research has laid the foundation for the flexible, robust, and precise motion needed for UXO clearing.

TABLE OF CONTENTS

I. INTRODUCTION ... 1
  A. MOTIVATION AND BACKGROUND ... 1
  B. OBJECTIVES ... 1
  C. ORGANIZATION ... 4
II. SHEPHERD SYSTEM DESIGN ... 5
  A. SYSTEM OVERVIEW ... 5
  B. MOTION MODES ... 6
III. SHEPHERD SYSTEM HARDWARE ... 7
  A. OVERVIEW ... 7
  B. THE MOBILE PLATFORM ... 7
    1. Motors and Encoders ... 8
    2. ServoController ... 10
    3. Gears ... 10
    4. Wheels and Tires ... 12
    5. Power Supply System ... 13
  C. SHEPHERD ON-BOARD COMPUTER SYSTEM ... 14
    1. Taurus Board ... 16
    2. Digital to Analog Board ... 18
    3. Digital Input Board ... 20
    4. Digital Output Board ... 20
    5. Counter Board ... 22
IV. SHEPHERD SOFTWARE SYSTEM ... 23
  A. OVERVIEW ... 23
  B. SOFTWARE ENVIRONMENT ... 23
    1. Bug Monitor ... 23
    2. S-Records ... 24
    3. Software Development System ... 26
  C. SHEPHERD REALTIME KERNEL (SRK) ARCHITECTURE ... 29
    1. Timer Control ... 29
    2. Motion Control ... 30
    3. User Interface ... 31
V. EXPERIMENTAL RESULTS ON MOTION CONTROL ... 33
  A. OVERVIEW ... 33
  B. WHEEL DRIVING ... 34
    1. Developing Driving Constants ... 34
    2. Measuring Wheel Speeds ... 37
    3. System Controls ... 40
  C. WHEEL STEERING ... 49
    1. Developing Steering Constants ... 49
    2. Measuring Wheel Rate of Turn ... 49
    3. Steering Feedback ... 50
    4. Wheel Testing ... 52
VI. MOTION MODES ... 53
  A. OVERVIEW ... 54
  B. "TORNADO" MOTION ... 56
  C. JOYSTICK CONTROLLED MOTION ... 59
  D. SEARCHING MOTION ... 62
VII. CONCLUSIONS ... 69
  A. SUMMARY ... 69

B. LESSONS LEARNED 70
C. RECOMMENDATIONS FOR FUTURE RESEARCH 71
APPENDIX A: SOURCE CODE (MAKEFILE) 73
APPENDIX B: SOURCE CODE (SHEPHERD.C) 77
APPENDIX C: SOURCE CODE (USER.C) 81
APPENDIX D: SOURCE CODE (MOVEMENT.C) 103
APPENDIX E: SOURCE CODE (MOTOR.C) 131
APPENDIX F: SOURCE CODE (TIMER.C) 165
APPENDIX G: SOURCE CODE (MATH.C) 171
APPENDIX H: SOURCE CODE (UTILS.C) 185
APPENDIX I: SOURCE CODE (SERIAL.C) 201
APPENDIX J: SOURCE CODE (CONSOLIDATED HEADER FILES) 211
APPENDIX K: SHEPHERD OPERATING MANUAL 239
APPENDIX L: SENSING SIMULATION DATA 253
APPENDIX M: SENSING SIMULATION CODE 265
APPENDIX N: INPUT VS. OUTPUT VELOCITY 283
APPENDIX O: INPUT VS OUTPUT STEERING RATES 287
LIST OF REFERENCES 291
INITIAL DISTRIBUTION LIST 293

ACKNOWLEDGEMENTS

This research was possible due to the gargantuan efforts of many people. Most notably is that of our thesis advisor, second reader, and Shepherd team members. Professor Kanayama's mentorship was truly instrumental to our in-depth understanding of kinematics and real-time systems. Professor Yun's ability to translate control systems theory into easily digestible layman's terms was invaluable. Without the efforts of team members Mike Williams and Thorsten Leonardi getting the project off the ground would have been nearly impossible—Mike's continuing maintenance efforts are a key factor in the "Shepherd" projects success.

We would also like to thank the faculty and staff members in the Computer Science Department, at the Naval Postgraduate School. Professor's Hensgen, Kidd, Lundy, and McGhee enhanced and broadened our knowledge base—even modified our Weltanschauung. Professor Luqi and the dining (hungry) philosophers provided a wonderful support network, helping us to focus on our goals. We also wish to thank the military faculty, Major Ludlow and Commander Holden for helping to ensure that our education was a value added prospect for the Department of Defense. Mrs. Valerie Brooks and Mrs. Cindy Holden should both be awarded medals for suffering through all the demos and constant noise from the robot.

Mrs. Jean Brennan and Mrs. Alice Lee of the CS curricular deserve great thanks for being a constant font of information, and for continuing to jump through all those last minute hoops with a smile.

Finally, We are grateful and indebted to our families their patience, understanding, and support throughout this research and our studies. Most of all for our life, health, and existence—Thanks be to the God of all creation!

I. INTRODUCTION

A. MOTIVATION AND BACKGROUND

Land mines are an inexpensive and effective defensive means in wars. The problem with land mines is that they remain to be a threat when wars are over. International efforts are being made to ensure that land mines deployed in the future are equipped with a time-out device, and mine locations are properly recorded. While such a treaty may provide relief in the future, millions of land mines were planted all over the world as a result of wars and regional conflicts in the past.

There are about 110 million land mines scattered around the world in more than 60 countries --- most of them in the Third World [Ref. 1, 2, 3, 4, and 5]. These land mines kill about 10,000 and injure another 20,000 people (many of them are children) every year. Moreover, there are millions of acres of the US formerly used defense sites (FUDS) that are contaminated with unexploded ordnance (UXO) as a result of military testing and training in the past [Ref. 6]. The contaminated land must be cleared inch by inch before transferring to civilian use. The difficulties of these clearing missions are in the variety of the objects to be identified and the diversity of the environments that are contaminated.

B. OBJECTIVES

As the military continues to downsize, the process of turning the land over to the civilian sector is sensitive, intensive, and costly. The aforementioned costs are both monetary and in some instance the loss of human life. One of the most complex problems is the clearing of UXO's from the FUDS.

The Department of Defense (DOD) has recently approved two organizational structures to confront the challenge of UXO remediation and wide-area de-mining. The objective of the first committee is to develop fully coordinated requirements driven research and development program for countermine, de-mining, site remediation, range clearance, and explosive ordnance disposal. Within the first committee there is a specific

group focused on detection technology. The second committee will focus on current technologies and ways to improve in the future. One of the phases will examine current UXO remediation, active range UXO clearance and explosive ordnance disposal efforts. Hence, the UXO problem is serious and a highly visible issue within the DOD. The current approach to mine and UXO clearing is dangerous and labor intensive [Ref. 6]. In a typical UXO clearing scenario, Explosive Ordnance Disposal (EOD) technicians walk slowly and carefully over a contaminated field in an attempt to identify the presence of UXO's that may be fully buried, half buried, or totally on the surface. UXO's found on the surface are visually examined to determine their types and fuse mechanisms. If fuse mechanisms are recognized and the condition of the UXO permits (e.g., not rusted, decayed, or encased in soil), an effort is made to defuse UXO's. UXO's that cannot be defused are gathered at a safe location and are destroyed using shaped charges. Transporting any live UXO is extremely dangerous and the motion of the transport vehicle must be gentle. Moreover, buried UXO's must be unearthed first. Therefore, UXO clearing consists of detection, identification, defusing, excavating, transporting, and disposal. Through robotics and the use of advanced technology it should be possible to make UXO clearing tasks safer, cost-effective, and more efficient.

At the Naval Postgraduate School a team has been put together to develop a semiautonomous vehicle or robot, which will survey possible contaminated areas for UXO's. The "rotary" class vehicle by its very design is capable of independent driving and steering with each wheel. Hence, rotary vehicles are capable of stronger torque and traction than most other vehicles on rugged terrains. Rotary vehicles with two wheels (on a very smooth surface) have been shown to possess the aforementioned enhanced torque and traction – with an increase in the number of wheels on the vehicle the capabilities are significantly improved. The vehicle needs to be highly mobile and capable of producing very fine motion to negotiate and search contaminated sites. To fulfill the requirement of the special wheel architecture, the semiautonomous vehicle, called Shepherd (a rotary vehicle), is presently under development. Shepherd possesses stronger torque and traction on rugged terrain, because of its special architecture (i.e., useful for the

mine/UXO. Mission). The name "Shepherd" was given to the vehicle because of the protective function of the "shepherd" in many ancient cultures—hopefully this Shepherd will also save lives. Shepherd (Figure 1.1) is a four-wheeled vehicle with independent steering and driving capabilities. The four-wheel independent driving and steering capability provides Shepherd a high level of mobility and preciseness in motion control [Ref. 7].

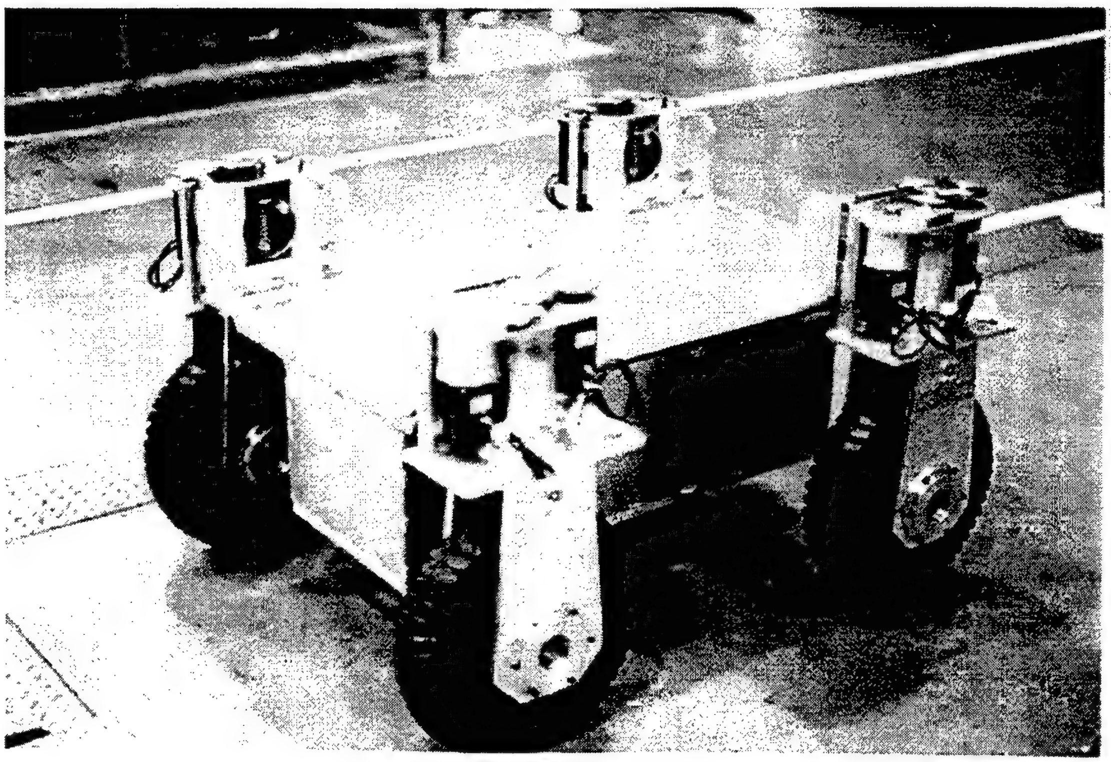

Figure 1.1: A "rotary" vehicle: Shepherd.

The fundamental objective of this thesis research project is to assist in the construction of a user-friendly and high-precision semiautonomous robotics tool to help the unexploded ordnance (UXO) and mine clearing mission.

This thesis will examine the following research areas:

- What kinematics algorithms must be developed to support a vehicle with three degrees of freedom of motion? The aforementioned algorithms must support highly flexible, controlled, and precise motion.
- What types of controls are required to ensure the optimal mix of driving and steering resources? Moreover, what must be done to ensure that all the resources complement?
- How can the knowledge gained in the aforementioned research areas be used to develop searching motion?
- How should the hardware and software systems be implemented to support the aforementioned goals?
- How can human operators remotely control the vehicle through an appropriate interface?

C. ORGANIZATION

This chapter provides a general overview of traditional and current techniques for identifying unexploded ordnance. Chapter II provides the System Overview and illustrates the concept of motion control. Chapter III describes the Shepherd Mobile Platform and the On-Board Computer System. Chapter IV presents the Shepherd Software Description and places great emphasis on the Shepherd Real-time Kernel (SRK). In Chapter V the results of experiments and testing motion control are presented. Chapter VI explains the current motion modes and Chapter VII summarizes the thesis.

II. SHEPHERD SYSTEM DESIGN

A. SYSTEM OVERVIEW

In consideration of aiding the UXO task, what type of vehicle should be developed? This vehicle will face difficulties of clearing missions in the variety of the objects to be identified and the diversity of the environments that are contaminated. This vehicle has to have precise and smooth motion, display motion flexibility, and contain robust motion in varied environments such as soft soil and rough terrain.

The vehicle must be capable of precise and smooth motion while searching for UXO's. The very nature of trying to locate UXO's should be meticulous and cautious. Haphazzard and jerky motions could contribute to lost of the vehicle due to unwanted detonations. Also, motion flexibility is absolutely necessary. This will enable different approaches or techniques for locating UXO's. Finally, the motion exhibited must be robust and stable due to the nature of UXO environments. While traversing these environments, the vehicle should not lose its precise, smooth, and flexible motion characteristics. For these reasons, a rigid body vehicle with at least 2 steerable wheels capable of semiautonomous or autonomous motions and equipped with sensors for detecting UXO's was considered at the abstract level.

So, the rotary vehicle platform was chosen, with the addition of four steerable and drivable wheels and a powerful computer system for control. The four wheels have thick tires and each contains two motors, one for driving and one for steering. Because of this, three degrees of freedom motion is possible which allows for motion flexibility. The independently driven four wheels aids in providing stronger traction than any other wheeled vehicle allowing for the negotiating of uneven slopes , soft soil, or rough terrain. The vehicle itself also provides the capability for further expansion of the system which will give it the full capability of fulfilling all aspects of the UXO mission. Chapters III and IV expand on the details of the system architecture regarding the hardware and software of this rotary vehicle.

B. MOTION MODES

Due to the uniqueness of Shepherd's independent 4-wheel motion of 360 degrees, several modes of motion are possible. The possible vehicle motions are:

- Tangential -- the vehicle's change in direction of movement is equal to its change in heading.
- Constant orientation -- the vehicle's heading is constant regardless of the change in the direction of movement.
- Complex which falls into neither of the above.
- Rotation

At this stage of development of Shepherd the constant orientation, complex, and rotation motion modes have been implemented with work proceeding on the tangential motion mode.

The user interface menu has a list of specific motions designed for the vehicle, which encompasses the typical vehicle motions. This list includes 1-Stop, 2-Straight motion (autonomous), 3-Straight motion by joystick, 4-XY-motion by joystick, 5-Rotate, 6-Sinusoidal, 7-Tornado (external), 8-Tornado (internal), 9-Tangential, 0-Exit, a-Tangential motion II, and t-Test motion. The sinusoidal motion is an implementation of the constant orientation while the tornado motions are an implementation of the complex vehicle motion. Both tangential motions are an attempt to implement the tangential vehicle motion. Further coverage will be given to several of these motions in more detail.

III. SHEPHERD SYSTEM HARDWARE

A. OVERVIEW

The Shepherd system hardware consists of the mobile platform and the shepherd on-board computer system. The mobile platform is the "mechanical" part of Shepherd, which provides motion and is directed by the on-board computer system. The Shepherd on-board computer system provides the computing power required controlling and directing Shepherd. Figure 3.1 provides a global perspective of the Shepherd System Hardware. Shepherd has four wheels, which are controlled independently. Each wheel has two motors; one for driving the wheel and the other for steering. Moreover, the steering capability for each wheel exceeds 360 degrees. The maximum driving speed (determined empirically) is approximately 87 centimeters/second. The unique mix of driving and steering capability is what provides the challenge and motion flexibility of this vehicle.

Shepherd has a mass of 150 kilograms and is built to form a square chassis (frame). Shepherd's wheels are centered on the corners of the square leading to very elegant calculations, as you will see later. Shepherd's Alternating Current (AC) electric motors are powered by 12 batteries, which are charged by an external AC source through converter. Figure 3.1 is also a transparent view of Shepherd from above which shows the Central Processor Unit (CPU), input/output (I/O) boards, servoamplifiers, batteries, and wheel unit assembly.

B. THE MOBILE PLATFORM

The mobile platform consists of the vehicle body. The vehicle body includes the vehicle's frame, motors, encoders, servocontrollers, gears, wheels, tires, and power supply. The on-board computer system is not considered part of the mobile platform.

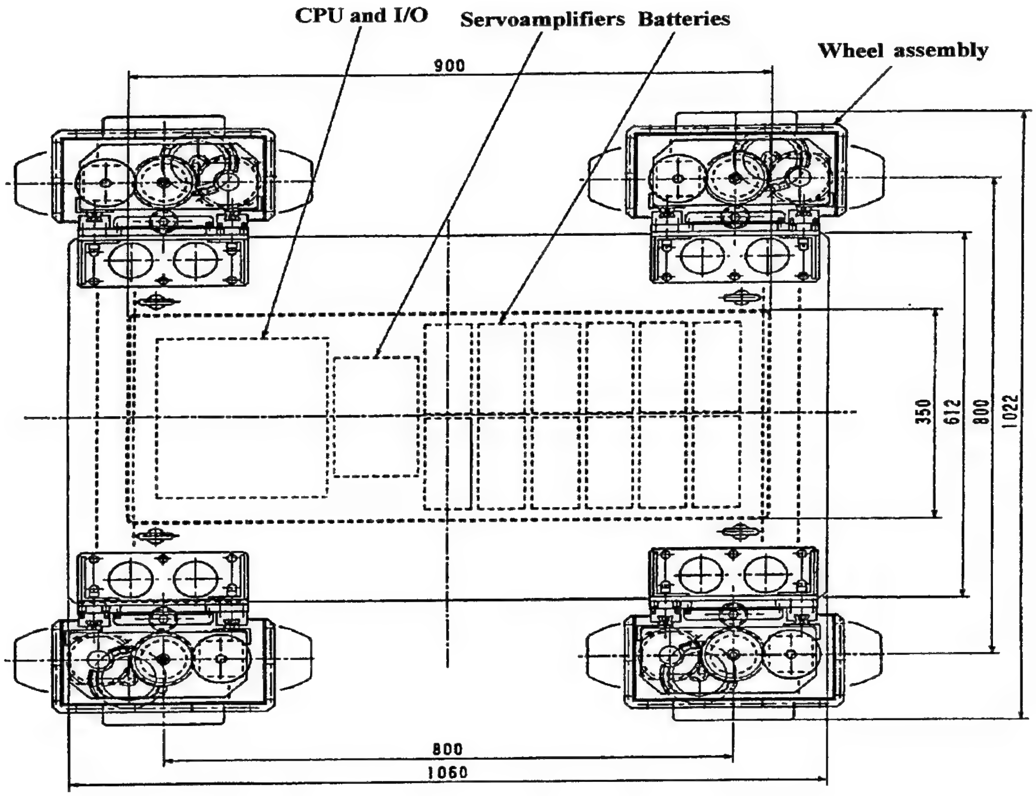

Figure 3.1: A transparent view of Shepherd from above. Showing the "relative" location of the CPU, and I/O boards, servoamplifiers, batteries, and wheel assembly. The relative position of the vehicle components is subject to change as the vehicle is modified.

1. Motors and Encoders

The eight motors and their corresponding shaft encoders used in Shepherd are from Yamayo Electric, Inc. These motors allow Shepherd to reach a "theorectical" maximum driving speed of 4 kilometers per hour (km/h), and a rate of 1 revolution per second about the steering axis (Ref. 7). Figure 3.2 provides the characteristics for the driving and steering motors.

Servomotor Characteristics		
	Driving Motor	**Steering Motor**
Nominal Torque	1.274 N-m	0.32 N-m
Maximum Torque	3.84 N-m	0.98 N-m
Nominal Rotation Rate	3000 rpm	3000 rpm
Maximum Rotation Rate	4500 rpm	4500 rpm
Size	60 X 123.5 mm	54 X 86 mm
Weight	1.7 Kg	0.74 Kg
Power (AC)	400 W	100 W

Figure 3.2: Servomotor characteristics for Shepherd.

Figure 3.3 illustrates the relative motor (M1-M7) position on the vehicle and the general orientation of the vehicle (e.g., front).

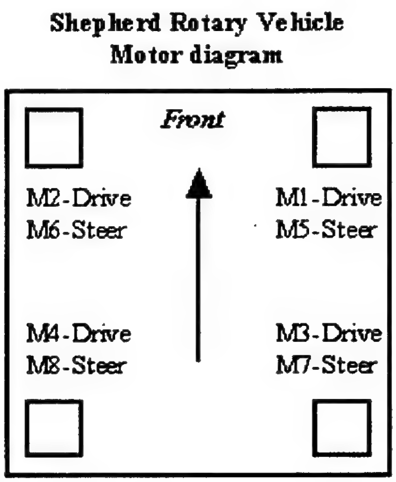

Figure 3.3: Shepherd motor diagram

2. Servomotor Controllers

The servomotor controllers actually provide the commanded voltage and current to the driving and steering motors to effect motion. The importance of the servomotor controllers can not be understated; the values written to the digital output board (and read from the digital input board) are within the acceptable range of the controllers. Also, the interface specification matches the range for the motors used on Shepherd and input signal voltage corresponds to the range allowed for the driving and steering motors. Later sections of this document will show that the voltage produce by the batteries is approximately 144 volts, which is within the acceptable range of the controllers. Figure 3.4 contains the characteristic and interface data for the servomotor controllers.

Servo Motor Controller Specifications	
Motor Capacity	400 W
Interface Specification	3000 rpm/5000 rpm
Output Current	8A
Control Method	PWM
Input Control Voltage	DC + 120~150 V
Input Signal Voltage	DC +/- 10V
Input/Output Signal	8 bit

Figure 3.4: Servomotor controller's characteristic and interface data.

3. Gears

Shepherd's reduction gear system contains flat gears, planetary gears, and bevel gears. This gear configuration has a 1:50 gear ratio for both driving and steering [Ref. 8]. Figure 3.5 provides a transparent view of the flat gears in the wheel assembly. Due to the gear configuration, when the wheels are used for steered then some driving is also initiated. The aforementioned driving is cancelled by applying the required amount of

"opposite" driving. And this "opposite" driving is based on the 1:50 gear ratio and is handled in the Shepherd code (Appendix J ,Consolidated header files, line 360).

Figure 3.6 is a side cut away of the wheel assembly. This cut away shows the gears involved in transferring force from the motors to the wheels. Gear ratio and force calculations can be obtained from [Ref. 8]. Also, mounted one of the flat gears is a "hall" sensor, which Shepherd uses to determine if wheel is aligned.

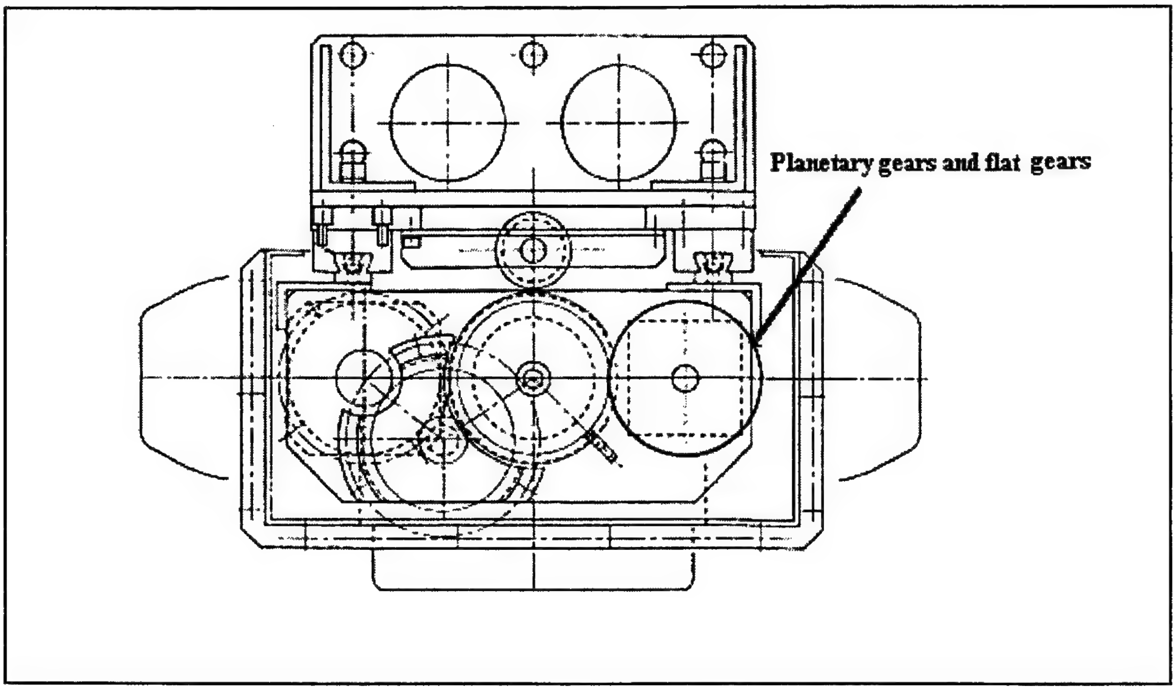

Figure 3.5: Flat gears in the wheel assembly (transparent view from the top)

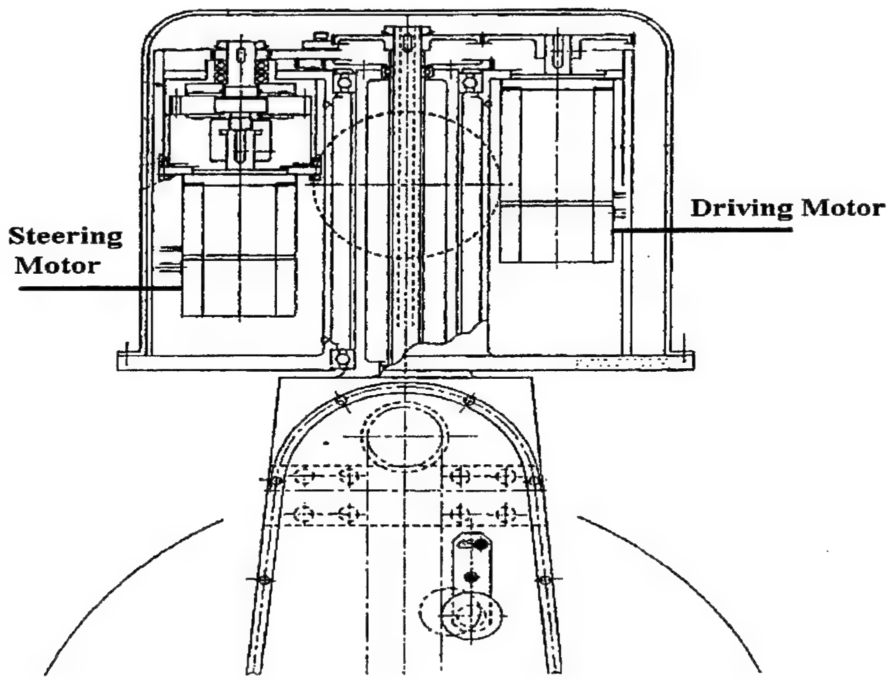

Figure 3.6: Side cut away of the motors and gears used in the Shepherd driving and steering mechanism (diagram not to scale).

4. Wheels and Tires

Shepherd's wheel diameter is 400 millimeters (mm), and the suspension travel range is 100 mm. Shepherd has somewhat ruggedized tires, with a tire friction factor of .5. The maximim tire pressure has been calculated as 49.8 pounds per square inch (psi); where 36 psi is currently used.

5. Power Supply System

Shepherd's power supply consists of twelve (12-volt) batteries connected serially. The voltage generated from the batteries is between 144-150 volts (within the servomotor specifications). The batteries have been used for periods up to two (2) hours without any noticeable degradation of performance. Figure 3.7 shows the switches required for the operational settings of charge, run (battery), run (external) and run (external/charge). The following are valid switch settings for Shepherd.

- Charge: 1 -SW-C OFF
  2 –SW-B ON
  3 –SW-A ON

- Run (battery): 1 -SW-A OFF
  2 –SW-C ON
  3 –SW-B ON

- Run (external): 1 -SW-B OFF
  2 –SW-C ON
  3 –SW-A ON

- Run (external/charge): 1 -SW-A ON
  2 –SW-B ON
  3 –SW-C ON

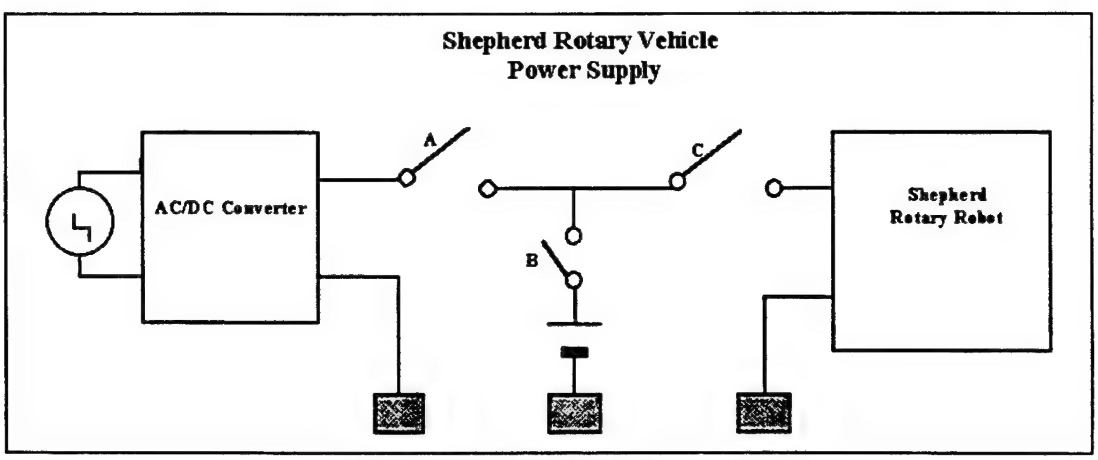

Figure 3.7: Shepherd power supply switch diagram

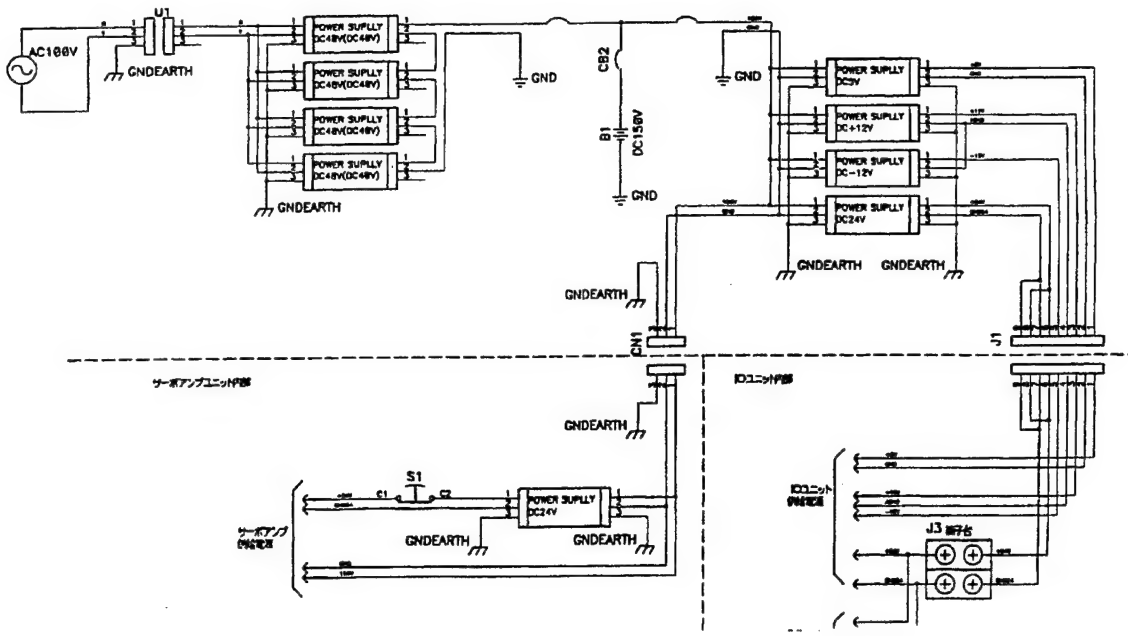

Figure 3.8: Simplified schematic diagram of Shepherd power supply. Showing both an external AC source and the 12-volt batteries serially connected.

An AC source (115-Volts) can be used to run Shepherd or to charge the batteries (accomplished by the AC/DC converter). Figure 3.8 is a global schematic of the Shepherd power supply [Ref. 9].

C. SHEPHERD ON-BOARD COMPUTER SYSTEM

The Shepherd vehicle's system design is illustrated by Figures 3.9, and is broken down into the hardware and software components both of which will be explained in greater detail in later. The hardware system is a combination of the mobile platform, an on-board computer system, servo drivers, batteries, and a laptop computer for a real-time I/O device. The computer system consists of a Taurus board housing two Motorola CPUs which are 68040 and 68030, a digital to analog board, a digital input board, a digital output board, a digital counter board; and a Versa Module European bus (VMEbus) based on Motorola architecture. Servo-controllers are connected to these I/O boards. Motor encoders are connedcted to the counter board.

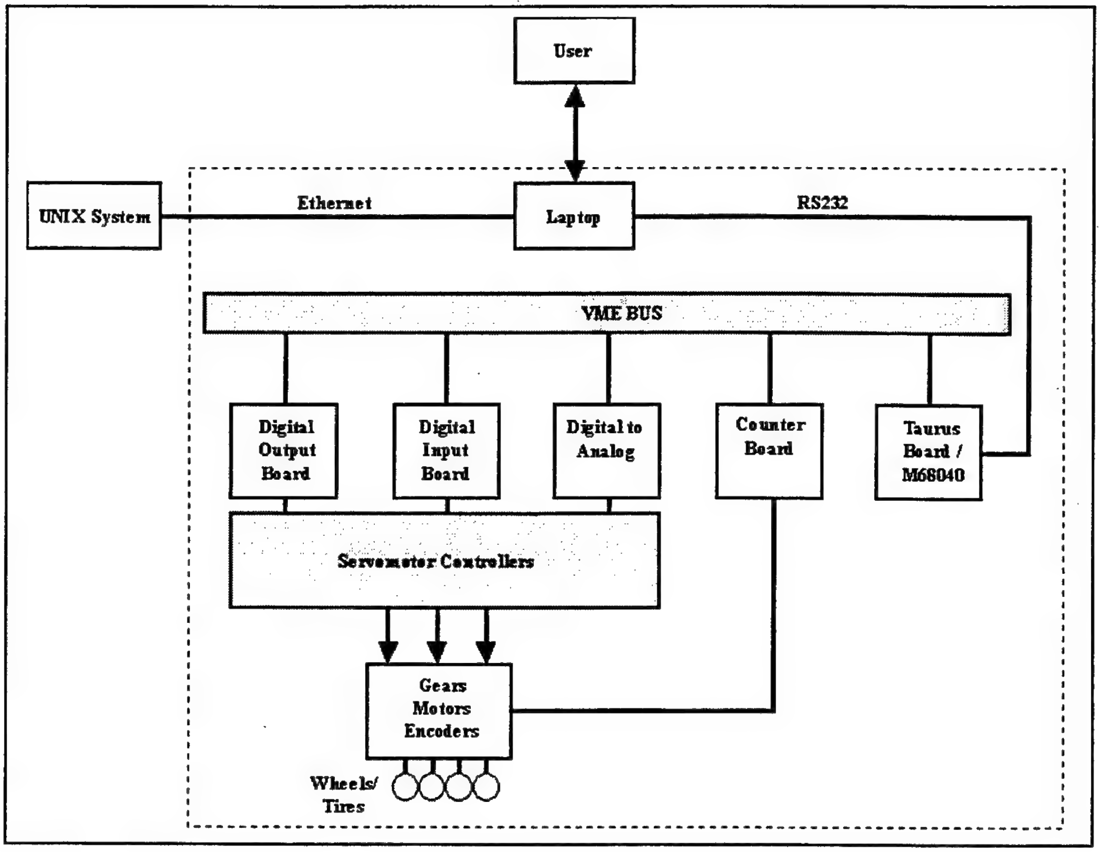

Figure 3.9: Diagram of the Shepherd on-board computer system.

1. Taurus Board

The Taurus is a dual-processor, and dual bus architecture, VME single slot, single board computer [Ref. 11]. The primary computing engine is a Motorola 68040 processor running at 25 MHz. The second processor on Taurus is the Motorola 68030. Although, taurus supports several real-time operating systems, an in-house operating kernel, SRK, was developed (chapter IV). Taurus also takes advantage of the direct memory access (DMA) functions provided by Ethernet, SCSI, and Intelligent Serial Controllers to DMA into main direct access memory (DRAM) through an isolation gateway between the M68040 bus and the M68030 bus. Moreover, Taurus acts as a fully functional VMEbus controller and may operate in Slot 1 of the VMEbus back plane (this is the case for Shepherd). Hence, the Taurus board is a powerful VMEbus engine and supports the requirements for a real-time operating system and a completely self-contained computing environment. Taurus features:

- 25 MHz M68040 Processor
- Burst Transfers
- Ethernet and SCSI with on chip DMA
- 16 Megabytes of DRAM main Memory
- 4 Megabytes of EPROM
- 1 Megabyte of Flash EPROM
- 25 MHz M68030 Processor
- 6 Serial Ports: 4 (RS-232-D Intelligent Ports with DMA), and 2 using a 68c681 device
- 32 Parallel I/O
- 11, 16 bit Timers (cascadeable into combinations of up to 80 bits)
- Interprocessor Mailbox
- Dynamic Bus Sizing
- Real-time clock with battery back up
- Watchdog Timer and 8 KB of battery back-up Static RAM

Why was the Taurus board chosen for this project? In addition to the aforementioned characteristics, the board uses the M68040 processor. In previous development of the Yamabico-11 [Ref. 12] robot, the M68020 was used. Motorola claims that its M68000 series chips are backward compatible. This research has concluded that this is mostly true, however on some key issues (math functions and assembly code) this has not proven to be the case. These issues will be revisited in chapter IV. Again a key asset of the Taurus board is how elegantly and logically internal and external interrupts are handled (Figure 3.10).

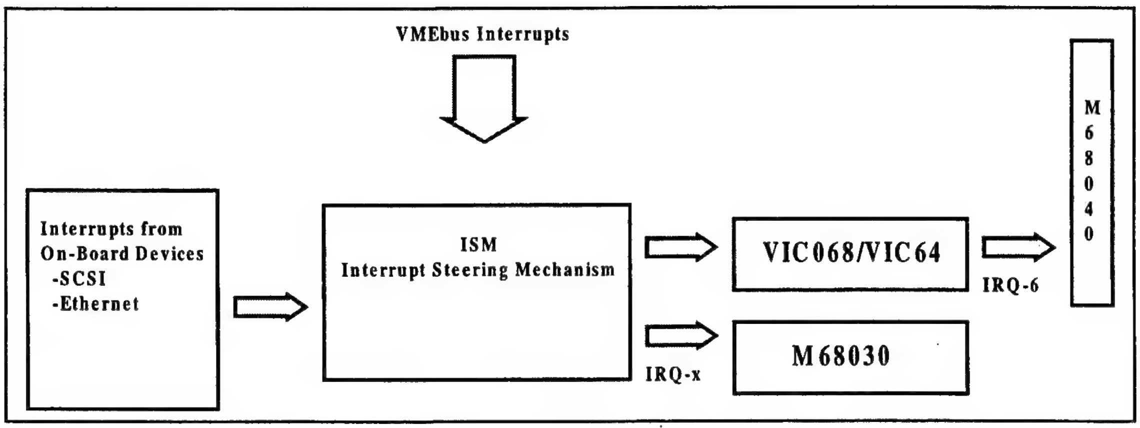

Figure 3.10: Interrupt handling diagram for the Taurus board [Ref. 11]

The Taurus board's communications facility adds flexibility to the implementation of RS-232 or Ethernet. The M68040 chip is a very versatile processor and powerful processor, and can perform 14 different operations at a given time. That is 6 operations by the Integer Unit (IU), 3 operations by the Floating-Point Unit (FPU), 4 by the Memory Management Unit (MMU), and a Bus interface operation. The number of timers on the Taurus board is also of a great benefit, moreover Timer 5 will be discussed in chapter IV. The M68040 is highly parallel, with a 6-stage integer pipeline, that when filled, will execute an instruction for every clock cycle. Moreover, each of the MMUs can accomplish a cache access and address translation concurrently.

2. Digital To Analog Board

The *Acromag* Series 9210 Analog Output Board (AVME9210) provides the means for connecting and driving analog circuits with outputs from the VMEbus for the Shepherd system [Ref. 13]. The board has 8 channels; each channel has a 12-bit resolution. A DAC per channel is used for signal accuracy. The DACs are set up to accept either straight binary or two's compliment data. The board has five programmable ranges for output voltages; however, +/- 10-Volts will be used with shepherd because it is a direct mapping to the maximums for the servomotors inputs. Characteristics of the AVME9210 are as follows:

- 12 bit output resolution
- individual DAC per channel
- 8 channels per board
- Byte or Word data transfers
- Power up reset
- Pass/Fail status indicators on the front panel

DAC Data Register	
Data Register	Base Address Offset
Channel 0	+82H
Channel 1	+84H
Channel 2	+86H
Channel 3	+88H
Channel 4	+8AH
Channel 5	+8CH
Channel 6	+8EH
Channel 7	+90H

Figure 3.11: DAC data register, 8 channels of output. Each channel is a digital to analog converter. A two byte address is reserved for each data register.

A single channel represents a driving or steering motor in Shepherd [Appendix E, Motor.c, line 228]. The status control register controls the pass/fail light. The Shepherd

code toggles the pass fail light in some instances to ensure the system and code are functioning properly. There is a memory location for the board status indicator flags and reset. This memory location is one byte in length and is located at base address +81H. During the early manual testing this status control register was used exclusively to accomplish resetting the AVME9210. Also, each of the aforementioned two byte DAC data registers (Figure 3.12) are set up as follows:

MSB										LSB					
D11	D10	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	X	X	X	X
12 bits												4 bits			
Bit DAC Data												Undefined			

Figure 3.12: DAC data register (16 bit). Most significant bit (MSB) and least significant bit (LSB). The 12 bits are a direct mapping of input values to the board, integer range [1023, -1024].

The AVME9210 is located at base address 0xffff0400 and is represented in the Shepherd code by the "label" VME9210.

Motor and DA Board Address Mapping		
	Driving Motor and Address	**Steering Motor and Address**
Wheel 1	**M1** **VME9210 + 0x0082**	**M5** **VME9210 + 0x008A**
Wheel 2	**M2** **VME9210 + 0x0084**	**M6** **VME9210 + 0x008C**
Wheel 3	**M3** **VME9210 + 0x0086**	**M7** **VME9210 + 0x008E**
Wheel 4	**M4** **VME9210 + 0x0088**	**M8** **VME9210 + 0x0090**

Figure 3.13: AVME9210 address to "physical" motor mapping.

3. Digital Input Board

The *Acromag* Series 9421 Isolated Digital Input Board (DIB) provides the means for connecting the Digital DC inputs the VMEbus for the Shepherd system [Ref. 14]. The DIB board isolates all digital inputs from the VMEbus for up to 250V AC, or 350V DC on a continuous basis (falls within the constraints of the Shepherd servomotors). The pass/fail light on this board is similar to the one used in the digital to analog board previously mentioned. And the DIB also has the input channel on light as well. The board has 64, 1 bit channels configure as four, 16 bit words. The inputs can be bi-polar (with polarity being +/- or -/+ at either end of the channel). The bi-directional polarity allows Shepherd to use this for changing the direction of wheel driving or the direction of steering with a change of input polarity. The DIB has the base address of 0xffff0000 and the "label" VME9421 is used in the code [Appendix J, Consolidated header files, line 386].

4. Digital Output Board

The Microsystems International Corporation 32-bit Optically Coupled Digital Output Board (VMIVME-2170A) consists of VMEbus compatibility logic, data output control logic, four 8-bit output registers, and 32 bits of isolated outputs. The VMEbus logic contains address decoding logic and data transfer control logic, which provides for 8- or 16- bit data transfers in the "short" I/O address space. The data output control logic selects byte or word transfers to the 32 optically isolated channels. The Shepherd research group spent many hours attempting to master this logic—a key problem was how to determine where the least significant value was for each data register. However, Thorsten Leonardy's efforts paved the way for a consistent and logical method of writing to the aforementioned registers. From this the Shepherd group was able to selectively choose combinations of motors for steering and driving [Appendix J, Consolidated header files, lines 181-192]. Figure 3.14 shows the register bit definitions [Ref. 15].

$XXX0 Data Register 0							
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Output Data							
OD31	OD30	OD29	OD28	OD27	OD26	OD25	OD24
$XXX1 Data Register 1							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Output Data							
OD23	OD22	OD21	OD20	OD19	OD18	OD17	OD16
$XXX2 Data Register 2							
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Output Data							
OD15	OD14	OD13	OD 12	OD11	OD10	OD9	OD8
$XXX3 Data Register 3							
Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Output Data							
OD7	OD6	OD5	OD4	OD3	OD2	OD1	OD0

Figure 3.14: Register Bit Definitions

The output board has the base address of 0xffffff00, which is represented in the code by the "label" VME2170.

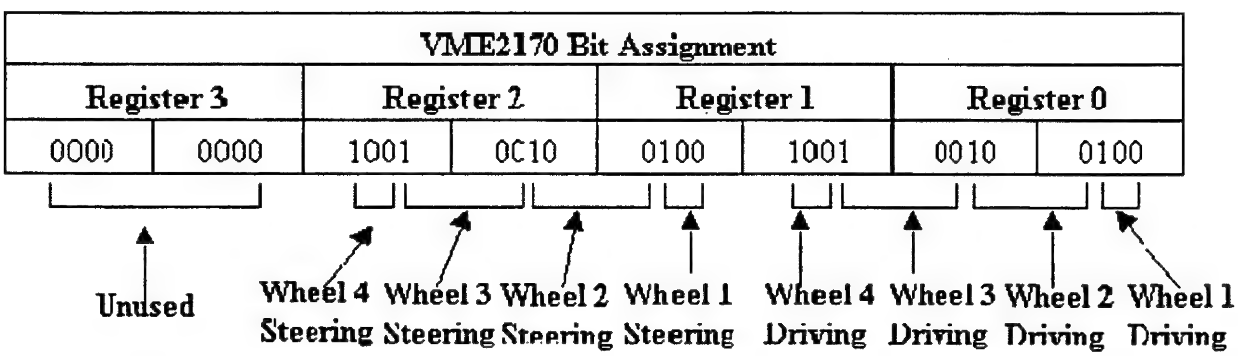

Figure 3.15: VME2170 bit assignment. Note 3 bit assignment for driving or steering motor selection.

Figure 3.15 indicates that "masks" could be written to the VME2170 base address and used for motor selection. Hence, writing a mask for 0x00000004 would select, motor 1 (M1) in wheel 1 for driving. And writing a mask for 0x00004000 would select, motor 5 (M5) in wheel 1 for steering. Moreover, using a mask for 0x00924924 all motors can be selected (using a "logical And" on the true values for each motor selected).

5. Counter Board

Shepherd uses the Green Spring IP-Quadrature Four Channel Quadrature Decoder or counter board [Ref. 16]. The counter board reads the signal produced by the encoders (see section III.B.1 of this document); this signal is "index" pulse once per revolution to provide absolute position information. There are four channels on the counter board; each channel has three inputs. The inputs are normally called X, Y, Z, and the board inputs are X and Y. Z is the control or index input. Each channel has a 24-bit up/down counter block, 24-bit capture/match register and a 24-bit output latch allow for an accurate "on the fly" reading of Quadrature position values. The up/down count direction in the counter board is controlled by the relative phase of the X and Y inputs. Count direction can be reversed by: reversing the mechanical motion direction, reverse the connections for X and Y, reverse the X polarity bit, or reverse the Y polarity bit. Shepherd uses a function called "readEncoders" which is an excellent example of completing three consecutive 8-bit reads from the counter board and catenation the 8-bit segments into a single 24-bit position reading [Appendix E:, Motor.c, lines 300-356 and 606-620]. The counter board's base address address is 0xffff6000, and is represented by the label VMECTR1 in the Shepherd code.

IV. SHEPHERD SOFTWARE SYSTEM

A. OVERVIEW

The Shepherd software system consists of the software development environment (including GCC version 2.7.2.1 compiler), Shepherd Real-time Kernel (SRK), and the firmware on the Taurus board. Shepherd software is developed in the "C" language on a Unix workstation, and the code is cross-compiled using switches with the GCC compiler to ensure viability on the Taurus board (step by step instructions are in the Shepherd Operating Manual, Appendix K). The code is in "S" record format and is transferred (via Ethernet) to a laptop computer, which is used as the user interface. When the user is ready to test or run a program, it is then download via RS232 to the RAM on the Taurus board using the Taurus bug or firm-ware on the Taurus board. The SRK has real-time timer control. The timer interrupts are set for 10 milliseconds, but can be modified to suite user needs and requirements. SRK's central motion control sections are shepherd.c, user.c, and the driver routine (all of which will be discussed in great detail later). The user interface viewed from the laptop computer is also generated from the code in user.c and other I/O code segments. The overall software environment and firmware works together to form a tightly coupled and low overhead "operating kernel" that is the SRK. The use of the SRK allows the user to control (i.e., timer interrupt, motion control, and user interface) the Shepherd rotary vehicle.

B. SOFTWARE ENVIRONMENT

The software environment (minus the SRK) includes the following: Bug monitior, S-records, and Software development environment.

1. Bug Monitor

The "taurus bug" is a powerful debugging and evaluation tool, and is firmware on the Taurus board [Ref. 20]. It has facilities for loading and executing user programs under complete operator control and evaluation (and is used extensively in Shepherd development). The "taurus bug" includes commands that allow the user to display memory, modify memory, set and remove break points, an assembler/disassembler, and a

system self test capability that verifies system integrity upon power up. The "taurus bug" also has various routines to handle some I/O, string functions, and data conversion via the TRAP #15 handler on the Taurus board (used in Appendix C, lines 535-548). Moreover, on power up, all static variables are set to default states, the break point table is cleared, all target registers invalidated, I/O character queues cleaned, the vector interrupt table is written to RAM, and all on-board devices (serial ports, timers, etc.) are cleared or reset. Taurus bug also has a system reset and abort feature. The system reset completely re-initializes the board and the abort feature captures a snapshot of the processors present state—allowing the use of stack pointers, and the program counter to help determine errors (the hardest way to debug). The "taurus bug" was a very valuable tool; however, at times it was difficult to use. And tracing through assembly code to resolve a problem using the "Trace" function and the symbol table can take numerous man-hours –without immediately yielding a positive result. A very important function of "taurus bug" is the loading capability. The use of the "Lo" command to place executable code in memory is key in the development process. The "S" records are downloaded to the Taurus board in this manner.

2. "S" Records

The S-record format was devised by Motorola for output modules. Its key purpose was encoding programs or data files in a printable format for the transportation between computer systems. Hence, providing a way of visiually monitoring the transportation processs and a method of quicky editing the code if required. S-records are character strings made of several fields which identify the record type, record length, memoery address, code data and checksum (see Figure 4.1).

Field	Printable Characters	Contents
Type	2	S0-S9
Record Length	2	Character pairs in record minus type
Address	4, 6, or 8	The 2, 3, or 4 byte address at which the data is loaded into memory

Field	Printable Characters	Contents
Code/Data	**0-2n**	**From zero to 2n bytes of executable code, descriptive information, or loadable data.**
Checksum	**2**	**Least significant byte of 1's complement of the sum of the values represented by the pairs of characters making up records length, address and the code/data fields. Used for error checking**

Figure 4.1: S-record content chart [Ref. 20].

S-records module may contain the following types (and many more): S0 (header data), record containing address where code is to reside in memory (S1, S1, or S3), S5 (the number of records transmitted per block), and the termination record (S7, S8, or S9). A typical S-record might look like this:

```
S00600004844521B
S113000284F245F2212226A000424290008237C21
S11300100002000800082629001853812341001813
S9030000FC
```

A detailed byte wise explanation of S-records is contained in Ref. 20. The S-records for shepherd's development are generated during the linking and loading process. A special switch is used that allows the creation of a file named "shepherd.TXT". The "shepherd.TXT" file contains the S-records to be downloaded.

3. Software Development System

As mentioned in this section's overview the compiler used is the GCC 2.7.2.1. This compiler posed many problems for SRK development. One of the biggest obstacles was the passing of "composite" structures. Structure values had to be placed into dummy variables (Appendix D, lines 138-140) in order to get the code to execute. This was especially odd because the code compiles, but will not execute (if the dummy variables are not used). Also, several of the compiler switches that supposedly allow mathematical code (that was previously valid for the M68020 with math co-processor) would simply not compile. An inordinate amount of time was spent trying to resolve this issue because one of the initial precepts of the project was that old code from the Yamabico-11 robot system would be portable. There are still anomalies that work arounds were developed for. For instance, there is a "square root" function that compiles and works in every environment (other compilers) yet would never function when compiled using GCC 2.7.2.1. Moreover, because the compiler does not have libraries for I/O or standard math functions, these had to be derived to support the M68040 and Taurus board (again taking an inordinate amount time). The compiler did serve its purpose—because its freeware (the budget did not allow for commercial compilers).

One switch that did work was –m68040, which allowed the generation of M68040 specific code. The "makefile" makes use of the –m68040 (segment below):

```
shepherd.o : shepherd.c
        gcc -c -m68040 -o shepherd.o shepherd.c

timer.o : timer.c
        gcc -c -m68040 -o timer.o timer.c

user.o : user.c
        gcc -c -m68040 -o user.o user.c

motor.o :  motor.c
        gcc -c -m68040 -o  motor.o  motor.c
```

The early testing also required a complete understanding of the "**a.out**" generated from the assembler and the link editor (the link editor makes "**a.out**" executable files). The "**a.out**" (Figure 4.2) consists of: a header, program text, program data, text and data relocation information, a symbol table, and a string table (the last three sections may be omitted if the program is loaded with the –s option. All of the aforementioned information was useful because our earliest a.outs were in the wrong format, hence not useable by the M68040 due to using non-functioning compiler switches.

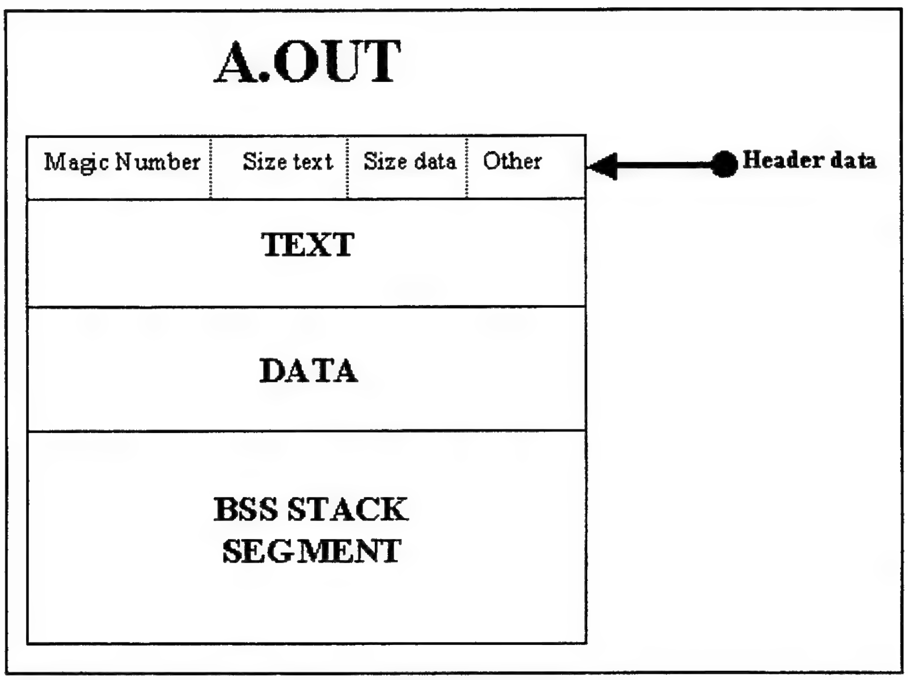

Figure 4.2: A generic A.OUT format. Note many features of the A.OUT are left off, such as symbol table, entry point, dynamic, and machine type.

The final element to be discussed is how the link editor is used. First, we must discuss the DRAM Memory map (Figure 4.3). The Taurus board documentation [Ref. 21] warns that accessing the memory below $10000 (Hex), hence the memory the format used in Figure 4.3 with 16 Megabytes as the upper bound.

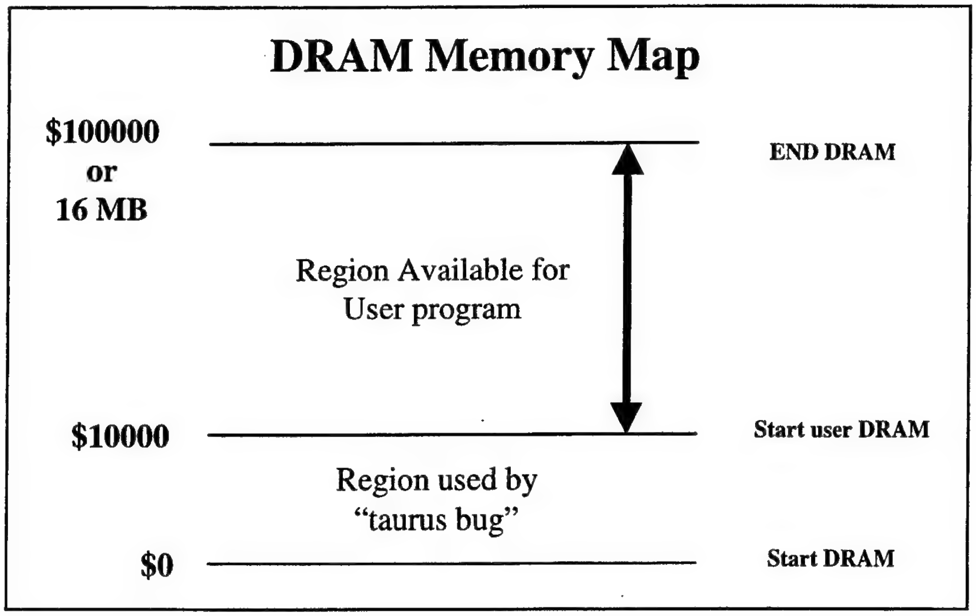

Figure 4.3: DRAM memory map. It should be noted that to be able to write to the DRAM the Parity ISM register must be disabled or the block- fill command (taurus bug) used to clear the region required [Ref . 12, and 22]. A detailed example is in the Shepherd Operating Manual (Appendix K).

Again the Shepherd makefile is illustrative of the linking process:

```
comp: startup.o shepherd.o timer.o serial.o math.o utils.o utils030.o user.o\
      motor.o movement.o

ld -Ttext 0x10000 -Tdata 0x20000 -Tbss 0x30000 -Map shepherd.map
      oformat srec\
      -o shepherd.TXT startup.o shepherd.o timer.o serial.o math.o utils.o\
      utils030.o user.o motor.o movement.o
```

The make file shows the text segment of the code being loaded at 0x10000, the data segment of the code loaded at 0x20000, and the upper bound for the data at 0x30000. Hence, the code is loaded within the parameters required by the memory map. Earlier it was mentioned that the S-records were generate by the linker—the "oformat srec –o shepherd.TXT" generates the required S-records for download to the laptop PC. If the "oformat srec" switch is not used then the standard **a.out** will be generated. The –Map

switch also allows the user to generate a symbol table (called shepherd.map here) for use in debugging. At this point all the underlying structure for SRK development is in place.

C. SHEPHERD REAL-TIME KERNEL (SRK) ARCHITECTURE

The SRK includes the Timer control, Motion control, and the User interface. Figure 4.4 below illustrates the exact architecture of the motion control part of the system.

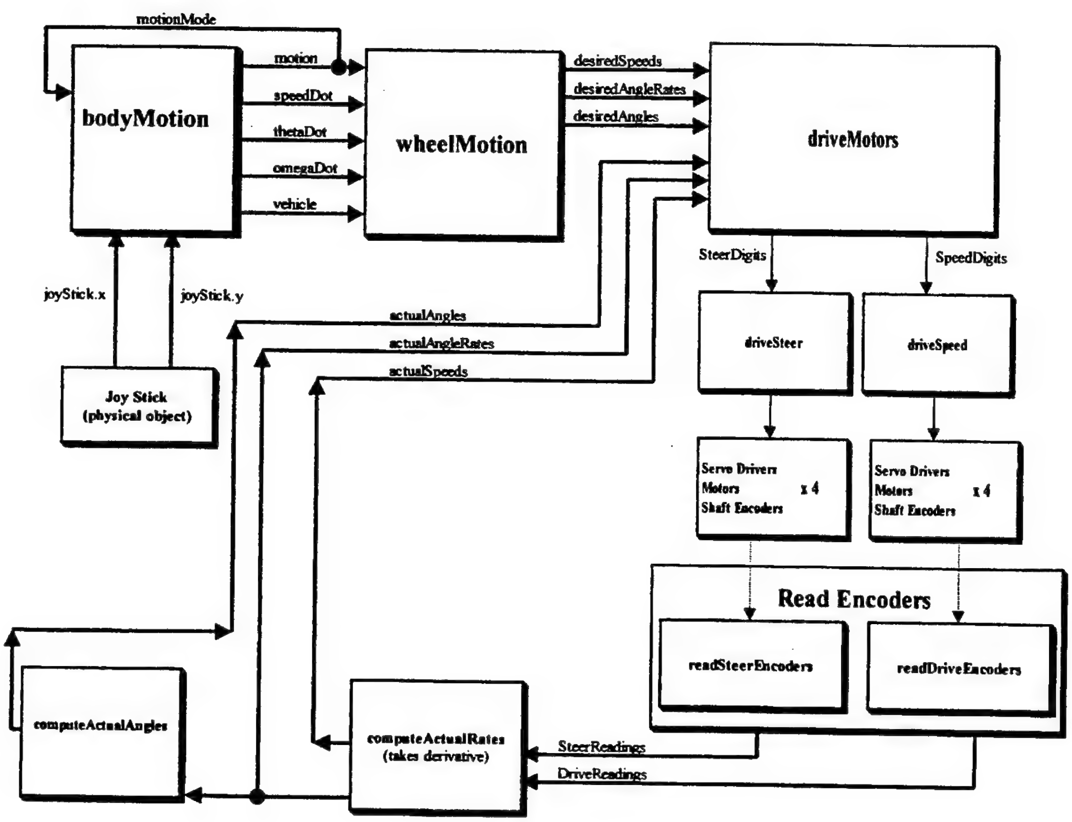

Figure 4.4: Shepherd Motion Control Architecture

1. Timer Control

The timer control is a very integral part of the system. As was stated earlier SRK has a real-time modifiable timer control. This timer control is made possible through the

use of the Taurus board's AM9513A Counter/Timer. This Timer/Counter provides five 16-Bit general purpose counters and uses a 4-Mhz oscillator as a clock input with the outputs connected to the Taurus Interrupt structure for processing. The Timer 5 group of the AM9513 device was utilized due to it's compatibility with the 68040 Processor and contains it's own timer handler.

The main routine in controlling the timing and setting the interrupts is located in the timer.c file along with the header file in timer.h (Appendix F). This timer is initialized and started in the main shepherd routine. It continuously provides a 10 millisecond interrupt until the program is terminated. However, this value could be modified. This was made possible by manipulating the data port values of the AM9513-1 device and multiplying those values by factors of 10000, 1000, 100, 10, or 1 to obtain a 1, 0.1, 0.01, 0.001, or 0.0001 second interrupt in that order. The accuracy of this timing was tested using an oscilloscope and also a frequency analyzer. It was found through testing that the SRK when in operation only utilizes approximately 2.7 milliseconds total in handling all the associated routines. For further information on this counter/timer see reference 21.

2. Motion Control

Referring back to Figure 4.4, motion control encompasses several major parts which are woven into a tightly controlled structure for driving and Steering Shepherd's wheels. The bodyMotion function takes as input the mode of motion desired from the user. Using this mode and the necessary instructions programmed, it provides 5 inputs to the wheelMotion function which are: motion, speedDot, thetaDot, omegaDot, and vehicle. Motion is a structure consisting of the user's input of speed, theta (direction of travel of the vehicle), and omega (rotational speed of the vehicle). Vehicle is a structure consisting of the x and y coordinate of the vehicle on some x-y plane for tracking purposes and the heading of the vehicle. SpeedDot, thetaDot, and omegaDot all are a derivation of speed, theta, and omega over time.

The wheelMotion function takes the given inputs along with inputs from a feedback loop for the actual positions of the encoders for both driving and steering of the vehicle, and performs calculations based on the theory discussed in chapter VI. The results of these calculations are then sent to the driveMotors function which provide these

values to the servo drivers for steering and driving the vehicle wheels resulting in the motion of Shepherd.

3. User Interface

The user interface is facilitated through the use of a Texas Instruments laptop computer running the Windows 95 operating system. By utilizing the HyperTerminal program accessory to connect the laptop to the unix system via an ethernet connection, program files can easily be receive from the unix system after being compiled and the executable files sent to the taurus board. After successful downloading of the program to the Taurus board, the user can then control the operation of Shepherd through the laptop keyboard. Figure 4.5 illustrates this user interface after the program is successfully downloaded and ran. An on-screen menu is displayed with a description of Shepherd motion that can be initiated via keyboard strokes.

This menu is located in the file user.c (Appendix C). It is a simple character definition which is called from the user.c file. The menu choice inputs from the keyboard are converted to ascii characters which are interpreted in the user() routine switch statement to launch the associated menu item seen in figure 4.5 on the next page.

Also, part of the user interface is a joystick. This joystick interface is activated when menu choices 3 and 4 are selected. It controls Shepherd's wheels for both steering and driving. This will be discussed in Chapter VI.

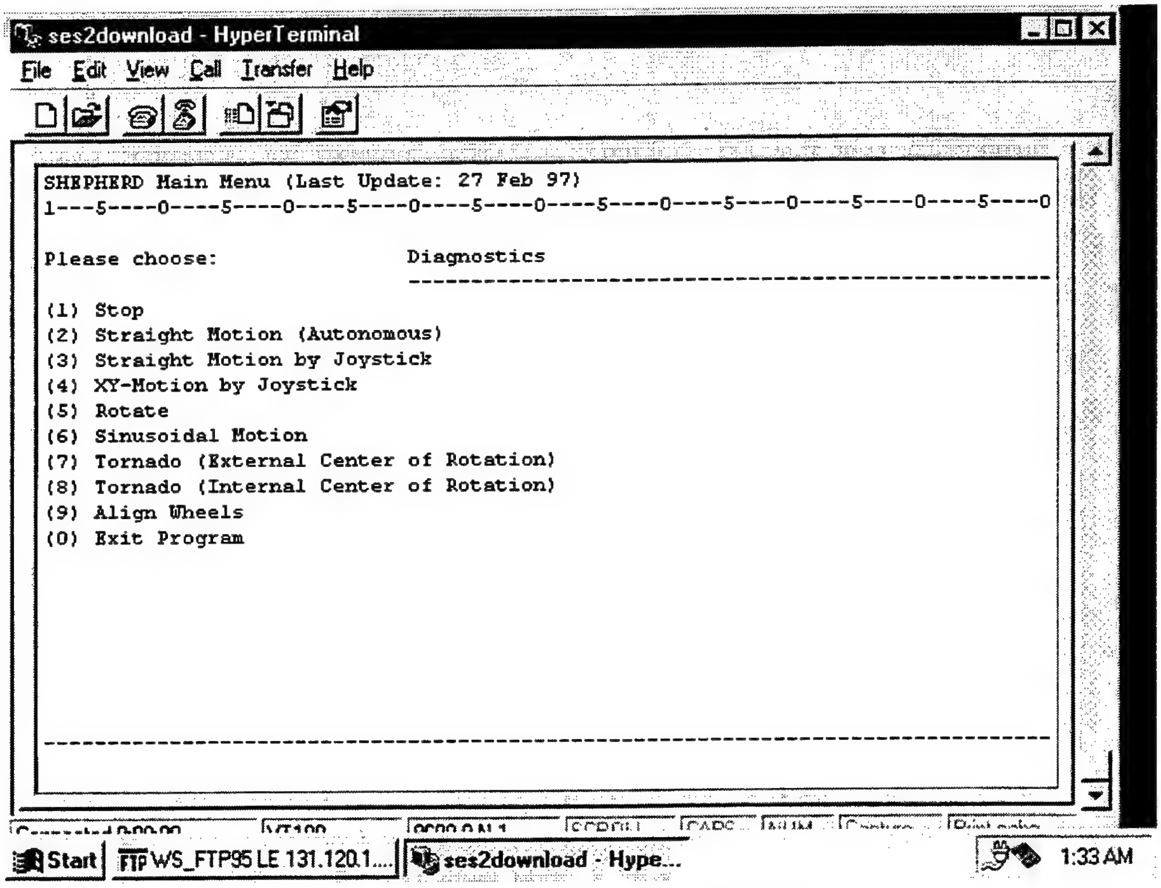

Figure 4.5: Display of shepherd menu for user interface.

V. EXPERIMENTAL RESULTS ON MOTION CONTROL

A. OVERVIEW

Chapters III and IV of this thesis discuss how each servomotor can be accessed and voltages applied. However, the aforementioned chapters do not explain how the incremental inputs placed on the DA translate to wheel speed or what the maximum and minimum wheel speeds are. So, how were the maximum and minimum wheel speeds determined? Moreover, what type "controls" are required to ensure that each wheel has the same driving velocity or angular velocity? We used a modified version of the "scientific method" to establish and carry out the experiments for the Shepherd vehicle.

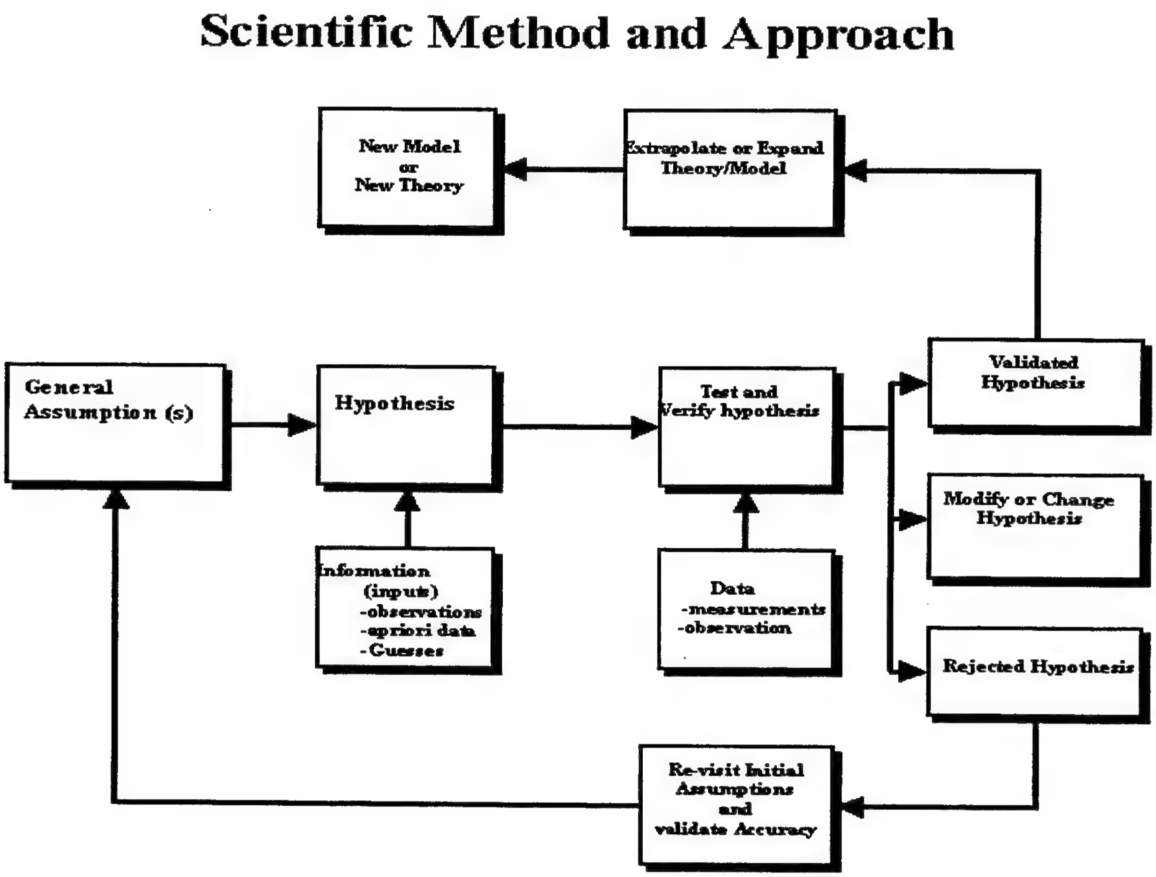

Figure 5.1: Modified version of the scientific method and approach used for Shepherd development.

As with any experiment, results must be consistent and reproducible. Considering the vehicle as a "closed system" observation of all experimental results should lead to deduction, connection, or correlation. As is common with the scientific method progress can be very rapid or slow. Sometimes there were unexpected results (e.g., singularities), and in some cases our dead-end path (failed experiment segment) provided some very important insights which helped to improve vehicle performance. An Ockham's razor approach was taken when making investigations and deductions—meaning that any unknown phenomena or behavior should be explained in terms of what is already known (and testing the simplest possibility first).

B. WHEEL DRIVING

Once an understanding of the mechanics and providing the coded structures to move the wheels had been achieved the concept of precise and controlled driving motion becomes the focus. The basic initial idea was to measure the counts from the counter board, measure wheel revolutions over time. Again reiterating a modified scientific method was used— and apriori data such as gear ratio and other engineering data were used to verify results.

1. Developing Driving Constants

The *digitToRadDrive* constant [Appendix J, Consolidated header files, line 364] was the first constant to be determined for wheel driving. The biggest problem with developing this constant was working around the 10-millisecond timer interrupt and the lack of a fully functional operating system. Since Shepherd's SRK was developed from the ground up within the last year it has limited I/O capabilities, and there is no long-term storage on the Taurus board. What this means is that the results of test programs would have to be sent to a monitor (VT220) or to a printer. We decided to print to both the monitor and printer. However, this also yielded unexpected results. The print function used too much of the 10 milliseconds to allow for proper functioning of the process currently running. Hence, we moved the print function from the real-time portion of the

code to alleviate this problem. Moreover, this became a useful technique that was used throughout Shepherd development and testing.

The first step in determining the number of counts is reading the counter board. In the algorithm used in the SRK, the initial values on the counter board are read, stored, and read again after a complete revolution of the tired is complete; the absolute difference between the initial counter value and the final counter is the actual count take for the revolution. It should noted however that consecutive reads of the counters is actually accomplished and the values adjusted for the transition from 0xffffff to 0x000000 and vice versa (24 bits) [Appendix E, Motor.c, lines 607-621]. An example of the code to read the encoders for driving can be seen below:

```
void readDriveEncoders(unsigned long int array[])
{
  unsigned char *p=(unsigned char*)VMECTR1, c1, c2, c3;
  int ix;
  long int temp;

  for (ix=0; ix<4; ix++) {   /* read all four motors subsequentially */

    *(p+3)=0x03;          /* load  output latch from counter */
    *(p+3)=0x01;        /* control register, initialize two-bit
                                  output latch */
    /* read three bytes for specific counter ix and save in status   */
    /* first access to Output Latch Register reads least significant */
    /* byte first                                    */

    c1 = *(p+1) & 0x00ff;
    c2 = *(p+1) & 0x00ff;
    c3 = *(p+1) & 0x00ff;
    array[ix] = ((unsigned int)c1) | ((unsigned int)c2 << 8) |
         ((unsigned int)c3 << 16);
   p=p+4;                 /* increment pointer for next counter */
  }
  return;
```

```
} /* end of readDriveEncoders */
```

Secondly, a method to display the enconder data is required. The SioOut [Appendix E:, Motor.c, lines 483-490 and 381-399] display routine from SRK was used for this, but failed because the time required to display the data exceeded 10 milliseconds. So, we moved the display "call" for the function from the real-time portion of the code. In SRK, immediately following the "driver" routine's call the following type routine would be placed in the main "user" routine:

```
while(1)
   {
   while(edCounter%200 != 0){};
      displayCount();
   };
```

The aforementioned routine would print based on the edCounter and the timer. The counter being incremented every 10 milliseconds. Hence the value is displayed every 2 seconds in real-time. The key is the fact that the printing is being executed outside of the driver routine.

Now all the elements are in place to determine the *digitToRadDrive* constant. After testing each wheel for a minimum of 1000 revolutions (called double pi or DPI below) the *digitToRadDrive* constant was determined:

```
#define  digitToRadDrive  -6.015495746e-5
            /* driving constant rad/count = DPI/104450    May 8 */
            /* Experimental Results by Ed Mays          May 7 */
            /* Wheel 1 count = 104456                     */
            /* Wheel 2 count = 104435                     */
            /* Wheel 3 count = 104454                     */
            /* Wheel 4 count = 104455                     */
            /* Average count = 104450                     */
            /* cf.  2048 * 51 = 104448                    */
```

Verification of the constant was made possible by the engineering data. Given the range of inputs (same as Figure 3.12) to the Servomotors and the gear ratio (section 3.B.3) the *digitToRadDrive* was be verified above (cf. 2048 * 51 = 104448). The value of 51 vice 50 for the gear ratio (was determined empirically to be 1:51).

The *digitToCmDrive* was much more trivial to determine. The *digitToRadDrive* provides the number of counts, knowing the wheel radius (18.9cm), and using circle circumference formula yields:

```
#define digitToCmDrive 0.0011369287
    /* driving constant cm/count = digitToRadDrive*18.9cm Ed Mays */
    /* 5/8/97                                                      */
```

Now having the "drive" encoder count allows the computation of distance traveled (cm) by the vehicle; and coupled with the timer interrupt allows for the computation of velocities and accelerations.

2. Measuring Wheel Speeds

The next step in control of the wheels is being able to manipulate wheel speed. Developing wheel speed control was one of the projects many dead-end path's—that eventually lead to great results. There were three problems with the work presented here. First, the inputs or digits were not applied at the lowest (hardware) level using SpeedDigits [Appendix E], instead the value was considered desiredSpeeds. Secondly, the input values were massaged before being used to ensure that all the system hardware would accept the range of values. Thirdly, the uniqueness of each driving motor and natural output variance over a range of inputs was not considered (e.g., used averages instead of individual motor data). How did this happen? A minor communication error and poor naming of arrays made this possible. However, a look at the results and the logic for deriving them will provide insight towards the actual solution.

Include are an estimated velocity and the software-measured (actual) velocity as determined by software. The estimated velocity was determined by applying the requisite input and measuring the number of seconds it took for wheel one to complete 10 revolutions (e.g., wheel radius 18.9cm, distance traveled = 2 * PI * Revolutions * radius =1187.5220 cm); inaccuracies from this measurement came from hand timing and the

version of pi used on a desk calculator. Velocity calculated from V=distance (cm)/time (seconds).

The software-measured velocity used the *digitToCmDrive* constant, which has the unit's cm/count. The algorithm subtracts the previous count from the present counter (as read from the counter board) and multiplies the result by *digitToCmDrive* leaving the outcome with the unit of cm. This outcome is then divided by .01 (DeltaT) representing 100th of a second or 10 milliseconds (corresponding to the system interrupt). The 99999999 represents a value not representable by the counter board. The output displayed to the monitor every 100 calls of the routine (mod 100). An example of the code to compute each wheel's driving speed can be seen below:

```
void computeActualRates()
{
 int i;
double count,speed;

 for(i=0; i<=3; i++)
  {
 if(PreviousCountSpeed[i] == 99999999)  /* for derivative for speed  */
  Drive_Speed_Actual[i]= 0.0;
else
  Drive_Speed_Actual[i]=
   (convertDifference((WheelDriveValues[i] - PreviousCountSpeed[i]))
   *DigitToCmDrive[i])/DeltaT;
 PreviousCountSpeed[i] = WheelDriveValues[i];
  }
}
```

Input (digits)	Time Stop watch (sec)	Estimated Velocity (cm/s)	Software Measured Velocity (cm/s, average)
10	116.27 sec	10.21349	10.23235
20	58.15	20.42170	20.46.471
30	39.10	30.37141	30.35599
50	No data	No data	50.70702
60	No data	No data	60.93937
70	No data	No data	70.94435
80	No data	No data	81.29040
90	No data	No data	87.65720

Figure 5.2: Inputs and results from massaged data (error). No data entries exist because the revolutions were too fast for hand timing.

If an input of 100 is used the estimated values for velocity are nolonger linear. If the inputs vs. velocities were linear the ratio 20/20.42170=100/X used to predict the estimated velocity would yield a velocity of 102.32355 cm/s; however, the software-measured velocity was 87.54350 on average for all input values from 100 to 1000 (Figure 5.3).

Input (digits)	Time Stop watch (sec)	Estimated Velocity (cm/s)	Software Measured Velocity (cm/s)
100	No data	No data	87.65720
200	No data	No data	87.65720
400	No data	No data	87.65720
1000	No data	No data	87.65720

Figure 5.3: Inputs vs. measured velocity. Figure showing a velocity saturation.

Hence, the relationship between the input and the velocity look somewhat linear until the area where the input is greater than 70-- after this velocity saturation seems to occur (or maximums reached). Moreover, if one closely looks at the slope (i.e., $y = mx + b$)

between the input ranges 20-30, 83-84, and 87-88 it is clear that the slope changes dramatically over these regions. Even though the wrong input structure and mechanism was used to generate this data a lot was learned. For instance, motor performance is not completely linear and each motor has somewhat unique characteristics. Once the aforementioned problems were identified and corrected the correct data [Appendix N] could be plotted. Figure 5.4 is a plot of the correct data.

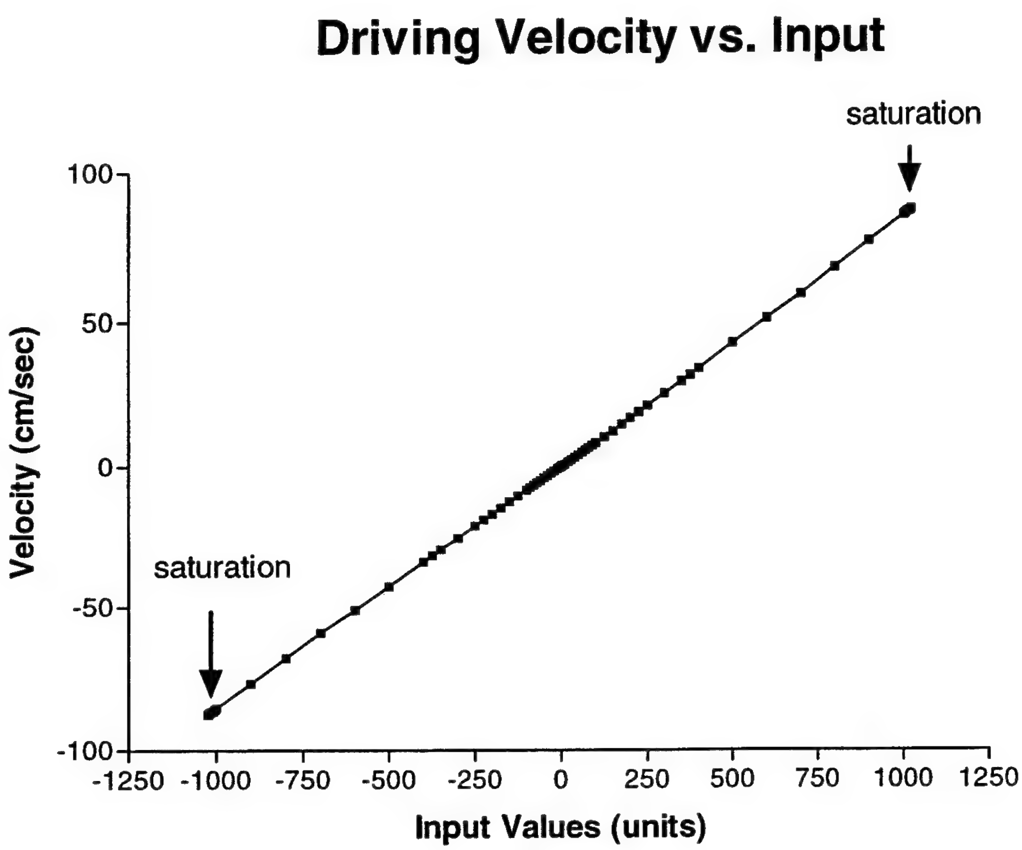

Figure 5.4: Driving Input vs. Velocity plot. Input units or digits are written directly to SpeedDigits. The slope changes for the overall graph are constant, making it look linear. However, over some regions this graph is non-linear, this will be dealt with under the controls section. Velocities also independently verified by use of tachometer.

3. System Controls

So far controlling the speed of individual servomotors has been discussed. However, it is known individual servomotors provide differing outputs for the same input

ranges in some cases. Moreover the wheels must be coordinated and work together. The goal of this project was to have an actual driving speed that has less than 1 % error when compared to the desired input speed (for each wheel). How is this close tolerance accomplished? This small margin of error is accomplished by using well-established concepts from "control systems" theory. A general control-system structure contains inputs (or reference commands), a controller (with external power), control forces, a controlled system (plant), disturbance inputs, outputs, and output monitoring. Control systems are almost a discipline unto themselves requiring knowledge of differential equations and Laplace transforms. Shepherd is looked at as a closed-loop where output monitoring is accomplished through sensors (encoders) and the information passed through feedback channels. The feedback results in a closed loop signal or information flow. The controller design for Shepherd is linear and considered a single-input-single-output (SISO) system. Hence, conceivably the state-variable or the transfer-function (input-output) method could be used here. What technique did we use? We used the trial and error technique [Ref. 17]. The trial and error technique (Figure 5.5) was chosen because of the skill and knowledge levels of the Shepherd team. The advantages of this technique are:

1. Simple mathematical tools are used
2. Vast amounts of experience have accumulated
3. Especially well adapted for use with computers
4. Linear designs usually are acceptable

The disadvantages are:

1. Inconsistent performance specifications (PS) can be encountered
2. Design is not optimal
3. The method is usually suitable mostly for SISO systems.

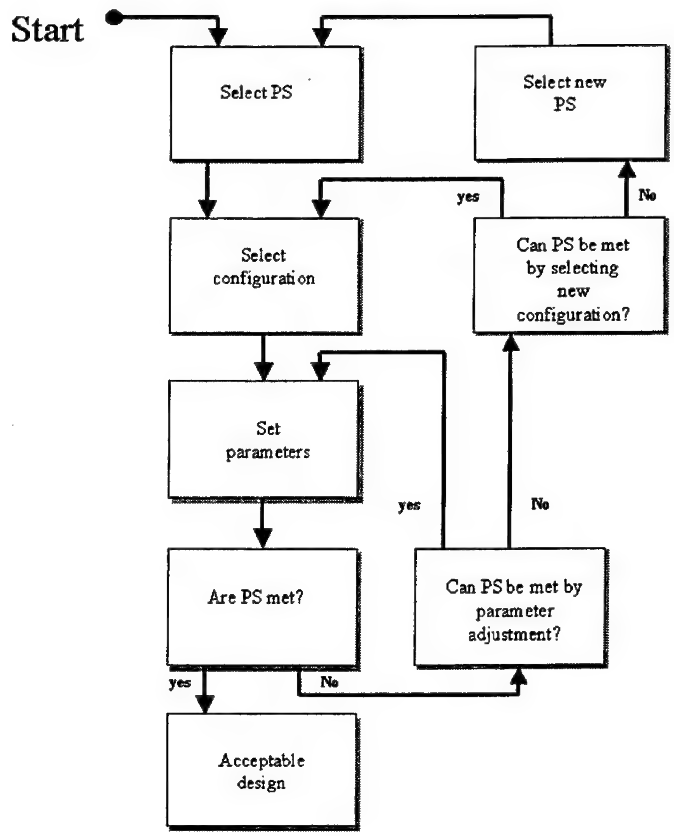

Figure 5.5: A flow graph of the trial-and-error design process [Ref. 18].

Using the experience of both Professors Kanayama and Yun as the guide the "black box" servo structure was developed (Figure 5.6). The servo structure is called "black box" because of the lack of understanding of the servomotors at that time (however, inputs and outputs could be measured). The previously mentioned PS was an error rate of less that 1% of the given reference input. There were other PS's governed by heuristics. For instance, it was desired that the actual output velocity converges (in the mathematical sense) on the reference or input velocity, with near perfect static follow-up.

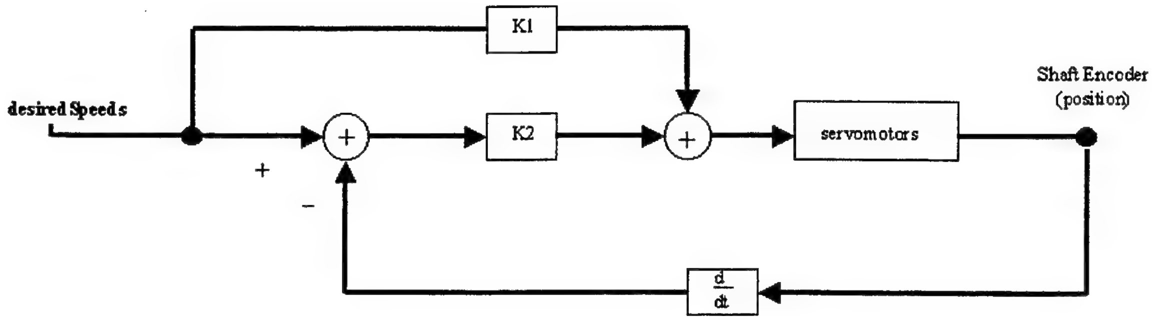

Figure 5.6: "black box" servo structure with feedforward and feedback compensation.

Vibrations and other unacceptable behaviors were used to determine if a smooth and acceptable convergence had been achieved for the gain used (e.g., if the robot was shaking, this was not acceptable). An acceptable gain would be one that produces an oscillatory response that converged quickly to the reference input (Figure 5.7). Again this same type information could have been determined by a better scientific guess [Ref. 19] using a closed-loop differential equation.

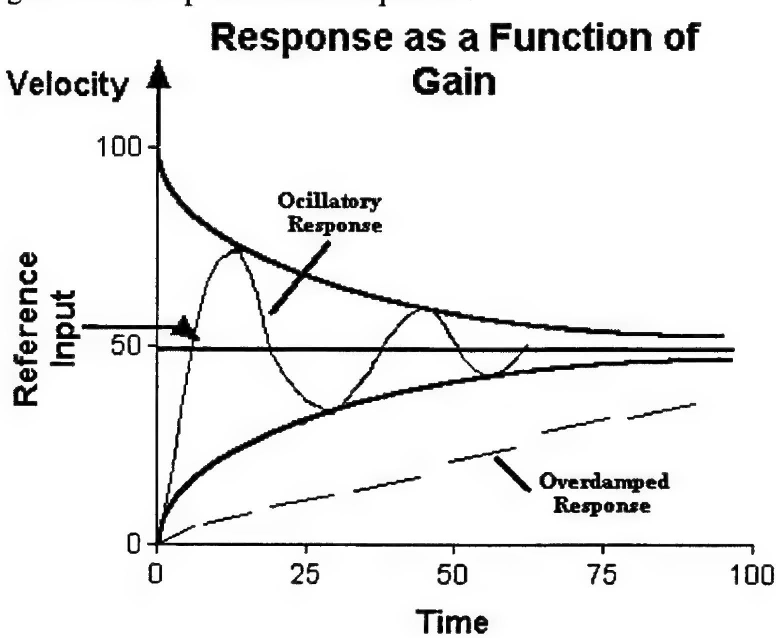

Figure 5.7: Proposed "proper" oscillatory response as a function of loop gain. Response determined for Shepherd using experience and heuristics.

Previously, it was stated that the input vs. velocity plot was not "truly" linear. These non-linear ranges have severe physical realities. Also, mentioned was the point that each servomotor produces an output that may not be the exact same as the outputs of the other system servomotors given the same input. And the stated PS requires that servomotor outputs be within 1% of the desired input or input speed. This translates into several challenges. First, using the feedforward technique constants were developed to ensure that the PS of 1% is met. The following algorithm or averaging technique was used for direct testing of inputs:

if $(v_1 < v < v_2)$

$d = d_1 + (d_2 - d_1)/(v_2 - v_1) * (v - v_1)$

The *v's and d's* above are the same as the velocities and inputs in figure 5.8.

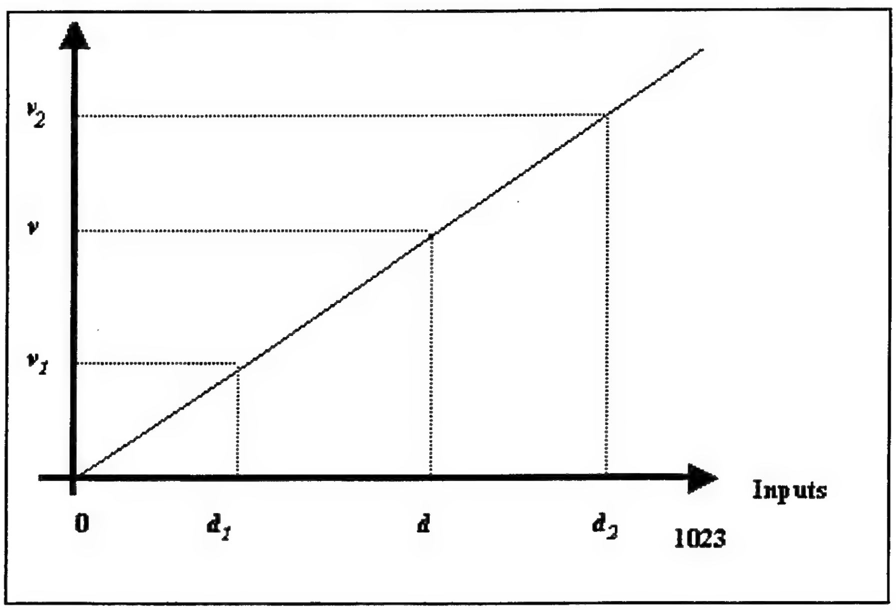

Figure 5.8: Illustration of averaging technique used to select "good" input ranges.

Using the aforementioned averaging technique the constants were developed that maintained all servomotor outputs within 1% PS.

These constants are called K1-K4, and K6. These constants represent the feedforward values that are applied to each servomotor over a specified range of desired speeds. The desired speed ranges give are because of the piecewise continuity chosen because of the non-linearity's in the input vs. velocity plot for the servomotors. The "K" constants were developed with no load on the system (e.g., wheels free floating). Listed below are the constants K1-K4, and K6 (speed is in centimeters/second):

```
K1[0]=11.448;/*0<=speed<=5,*/
K1[1]=11.500;
K1[2]=11.496;
K1[3]=12.375;

K2[0]=11.500;/*5>speed<8*/
K2[1]=11.500;
K2[2]=11.644;
K2[3]=12.000;

K3[0]=11.611;/*8>=speed<20*/
K3[1]=11.585;
K3[2]=11.686;
K3[3]=11.840;

K4[0]=11.711;/*20>=speed<=70*/
K4[1]=11.659;
K4[2]=11.705;
K4[3]=11.727;

K6[0]=11.710;/*70>speed<K5*/
K6[1]=11.700;
K6[2]=11.700;
K6[3]=11.715;
```

Above the bracketed values are array element numbers. Element [0] refers to wheel 1, motor M1; element [1] refers to wheel 2, motor M2; element [2] refers to wheel

3, motor M3; and element [3] refers to wheel 4, motor M4. A constant K5 was also defined as 87.4 cm/sec, and used a safety cutoff for the maximum speed. Figure 5.9 shows the values for a few selected desired or commanded speeds (most are within the require 1% error in PS).

Desired Speed	M1 Speed	M2 Speed	M3 Speed	M4 Speed
10	10.20	10.08	10.10	9.89
20	20.12	20.11	20.11	19.93
30	30.03	30.00	30.02	29.98
40	39.98	39.91	39.98	39.98
50	49.98	49.79	50.08	50.00
60	60.01	59.77	60.02	60.02
70	69.91	69.78	69.95	69.97

Figure 5.9: Desired (commanded) speeds vs. actual "free floating" motor speed. The application of feedback is expected to move the speed of M1 into the required 1% error for the PS.

At this point all the tools and techniques (i.e., trial-and-error, scientific method, and the experience of Professor's Kanayama and Yun) are in place for the application of system controls. How are the system controls (Figure 5.6) implemented? First, the algorithm is presented:

```
SpeedDigit = velocityReferenceTable(Omega_Speed,ix) +
        DriveFeedBackGain*(Omega_Speed - Drive_Speed_Actual[ix]);
```

SpeedDigit represents the actual digit value being applied to the servomotor. In the feedforward part of the loop (Figure 5.6) the commanded velocity is multiplied with a constant *K1*, which corresponds to the "*K*" constants described earlier in this section? In the code segment above the function "velocityReferenceTable" is called (prior to the addition of DriveFeedBackGain*(Omega_Speed - Drive_Speed_Actual[ix])). The "velocityReferenceTable" applies the proper "K" constant for the range the commanded speed (Omega_Speed) falls within.

A detailed look at velocityReferenceTable is provided below:

```
double velocityReferenceTable(double desiredVelocity,int i)
{
  double inVelocity,
       outVelocity;

  inVelocity=new_abs(desiredVelocity);

   if (inVelocity>=0.0 && inVelocity<=5.0)
    outVelocity = inVelocity*K1[i];

   if (inVelocity>5.0 && inVelocity< 8.0)
    outVelocity = inVelocity*K2[i];

   if (inVelocity>=8.0 && inVelocity<20.0)
    outVelocity = inVelocity*K3[i];

   if (inVelocity>=20.0 && inVelocity<= 70.0)
    outVelocity = inVelocity*K4[i];

  if (inVelocity>70.0 && inVelocity<K5)
    outVelocity = inVelocity*K6[i];

   if (inVelocity> K5)
    outVelocity=1023;

   if (desiredVelocity< 0.0)
    outVelocity = - outVelocity;

  return outVelocity;
} /* end velocityLookupTable */
```

It should be noted that if the inVelocity is greater than K5, then the outVelocity is given a value of 1023—this ensures there are no system resets because the input digits are too large. All other velocities are multiplied by a specific *"K"* and the value returned. In the feedback part of the loop (Figure 5.6) the difference between the commanded velocity and the actual velocity is multiplied with a constant *DriveFeedBackGain* (also *K2* in Figure 5.6) [Appendix E, lines 46 & 47]. Again this (DriveFeedBackGain*(Omega_Speed - Drive_Speed_Actual[ix])) is added back into the inputs used for the next time the process is run (based on the 10 millisecond timer interrupt).

Hence, now the final key to this control system would be finding a *DriveFeedBackGain* constant that would provide the desired "proper" oscillatory response (Figure 5.7) and ensure the servomotor output velocity is within the 1% error of the commanded velocity required by the PS. Using Occam's Razor, the trial-and-error flow diagram (Figure 5.5), the iterative approach in the scientific method (Figure 5.1), and the heuristics provided by Professor's Kanayama and Yun, the search for the best *DriveFeedBackGain* was initiated. The heuristics used were observation based. First, the "gain" used could not cause the vehicle to shake in any visible manner. Secondly, the "gain" used had to quickly move the actual servomotor speed to the commanded speed if there was a difference. Pseudo random values were chosen as gains, based on the experience of the aforementioned professors. The real number range [-1.0, 1.0] was used to test the gains. On the negative end of the range the gain was incremented by +.05, until the gain equaled zero—the results were not acceptable. On the positive end of the range the gain was decremented by +.05, until the gain equaled zero—at .8 the gain showed the best results (i.e., range [0.0, 1.0]). The gain was defined as *DriveFeedBackGain = .8* [Appendix J, Consolidated header files, line 389]. Hence the gain met all the criteria for an acceptable design (Figure 5.5) and validates the "black box" servo structure as envision by professors Kanayama and Yun (Figure 5.6). Moreover, the experiment can be considered a multidisciplinary success between physics, electrical engineering, and computer science.

C. WHEEL STEERING

As with wheel driving, the same approach to wheel steering was used in measuring the counts from the counter board and measuring the rate of turning of each wheel over time. The goal was to observe and measure each wheels turn rate and create a feedback compensation loop as in wheel driving to ensure that the PS of 1% is met.

1. Developing Steering Constants

The steering constants were developed in the same way as the driving constants. The counter board was read for steering values and then displayed . A digitToRadSteer (input digits per radians for steering) constant value of –6.817692391e-5 (rad/count = (2*pi)/(2048*45)) and RadRateTodigit constant value of 195.4155 (digits/rad/sec = 1023/5.23598) was determined by observation of the data forthcoming.

2. Measuring Wheel Rate of Turn

In measuring the wheel rate of turn, the same approach was taken as discussed before. An estimated rate of turn and the software-measured (actual) rate of turn are included. The estimated rate was also determined by applying an input and measuring the number of seconds it took for a wheel to completely rotate 360 degrees. As can be deduced, a certain amount of error was introduced due to human timing interaction.

Desired rate of turn (rad/s)	Time Stop watch (sec)	Estimated Rate (rad/s)	Software Measured Rate (rad/s, average)
1	6.0	1.00000	0.98174
2	3.5	1.79485	1.95667
3	2.19	2.86849	2.93160
5	1.69	2.71716	3.90653
5.5	No data	No data	4.88828

Desired rate of turn (rad/s)	Time Stop watch (sec)	Estimated Rate (rad/s)	Software Measured Rate (rad/s, average)
10	No data	No data	5.23598
20	No data	No data	5.23598
30	No data	No data	5.23598

Figure 5.10: Inputs and results from massaged data (error). No data entries exist because the revolutions were too fast for hand timing.

By observing the data from the above figure we can confirm that a certain amount of error is inherent in the system. As the input value increases from 0 to 5.5 radians per second , a linear correspondence tended to exist. However, after an input of 5.5 rad/s the software measured average tended to be 5.23598 rad/s resulting in a saturation state. This result in itself could not give us the exact range of values where this occurred. Therefore, manipulation of the steerDigits was necessary. Appendix O contains these results given in inputs of digits vice desired rate of turn. At the maximum input speedDigit of 1023 saturation is reached. The wheels will only turn at an average rate of 5.235 radians per second. Thus the software measured average of 5.23598 rad/s was adopted.

3. Steering Feedback

By further observation of the data obtained, an average was used in obtaining the rate of turn. Not all the wheels turn at the same rate. The objective was to have a less than 1% PS for optimization. To achieve that another "black box" servo structure was developed . The Figure 5.11 is a pictorial representation of this and is a little different than that of wheel driving.

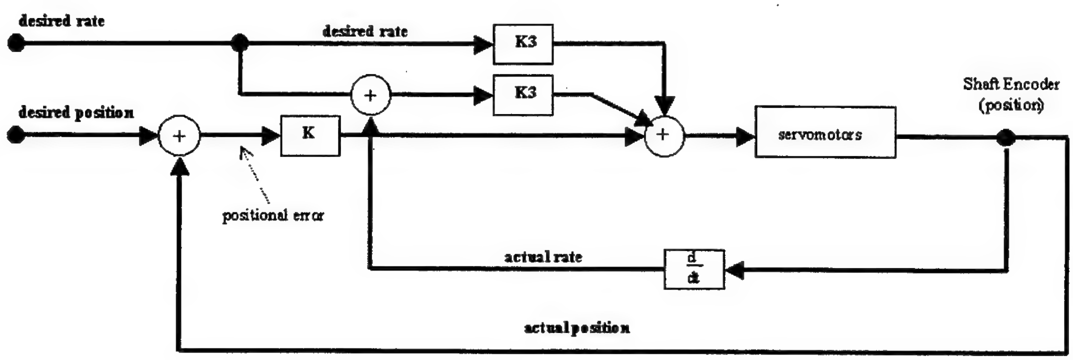

Figure 5.11: "black box" servo structure with series and feedback compensation for wheel steering feedback control.

In finding the K constants from the figure, trial and error is also used. The representation of the algorithm in code is presented:

```
Steer_Digit = rateReferenceTable(desiredAngleRates[ix])
              + steerFeedbackGain*(desiredAngleRates[ix]-actualAngleRates[ix])
              + angleFeedbackGain*norm(desiredAngles[ix]-actualAngles[ix]);
```

Steer_Digit represents the actual digit value being applied to the servomotor. The rateReferenceTable function simply converts the inputed rate to digits or, in the case where the inputed rate is larger than 5.235, clips it to within limits. The function is given below:

```
double rateReferenceTable(double desiredRate)
{
  double inRate,
         outDigit;

  inRate=new_abs(desiredRate);

  if (inRate<= 5.234)
    outDigit = inRate*195.4155 ;
  else
    outDigit=1023;
```

```
  if (desiredRate< 0.0)
    outDigit = - outDigit;

  return outDigit;
}
```

The two K constants steerFeedbackGain and angleFeedbackGain were determined by trial and error. AngleFeedbackGain was first maintained at 0.0 while testing steerFeedbackGain. SteerFeedbackGain was made very low at the outset and increased through each test until the vehicle displayed unusual behavior such as shaking while operating its steering function. At this point the value was lowered and then the value for angleFeedbackGain was increased in the same way. These values became optimal at 100.0 and 1000.0 for steerFeedbackGain and angleFeedbackGain respectively.

4. Wheel Testing

It was discovered while operating the vehicle that wheel 4 would on occasion be very badly misaligned from the other 3 wheels. Even after repeated realignment it would not operate as the others. At suspect was the thought this problem was software related . To test this, a routine was inserted into the SRK driveMotors() function with a menu item on the user interface. This routine simply turned wheel 4 360 degrees in one direction until the wheel aligned read the encoders for angle position, paused one second and then turned it in the opposite direction. At each pause, the wheel position was displayed to the interface screen and recorded. The data obtained is presented below in Figure 5.12 for 10 iterations for clockwise and counterclockwise rotation:

	Clockwise Rotation	**Counterclockwise Rotation**
1	**000.867**	**360.390**
2	**000.878**	**360.390**
3	**000.976**	**360.363**
4	**000.933**	**360.414**
5	**000.984**	**360.371**
6	**000.992**	**360.394**
7	**000.992**	**360.453**

	Clockwise Rotation	Counterclockwise Rotation
8	000.902	360.394
9	000.996	360.445
10	000.996	360.476

Figure 5.12: Wheel 4 data based on position of rest after direction of turn.

The average values for rotation in both the clockwise and counter clockwise directions were 000.955 and 360.426. These were averaged for 20 iterations see Appendix O for the full data set. Even though this data was obtained while the wheel was in a free floating environment without added friction, it still proved that wheel 4 was operating within 1% PS. Therefore, from this result a conclusion is drawn that the problem is not software related in nature but maybe mechanical.

VI. MOTION MODES

A. OVERVIEW

Chapter II mentioned several modes of motions that can be exhibited by Shepherd. Due to the nature of Shepherd's characteristics of having four wheels that can be independently operated with two degrees of freedom, this makes it possible to obtain three degrees of freedom motion. In this chapter we will discuss the modes of "Tornado" (complex motion), Joystick controlled motion and searching motion simulation. The other motion modes are encompassed by these motions. The code for the other motions are provided in the Appendices. Also, the Tangential mode will not be discussed but will be left for a future reasearch topic and implementation. However the ground work as well as the code is in place for implementation (Appendix D).

Before embarking on this discussion on motion modes the theory or basis for motion control must be presented This control motion theory, as well as the figures to follow, was taken from the works of Professors Yutaka Kanayama and Xiaoping Yun [refs 23 & 24]. First a vehicle coordinate system is defined on a rigid body robot. A configuration q is defined as

$$(p, \psi) = ((x, y), \psi),$$

where p is the positioning of the vehicle origin and psi is the heading orientation of the vehicle Xv-axis. Next in describing the motion of the vehicle's configuration which is a function of time, the following is the definition:

$$q(t) \equiv (p(t), \psi(t)) \equiv ((x(t), y(t)), \psi(t)),$$

where p(t) is the translational component and $\psi(t)$ is the rotational component of the vehicle motion. Figure 6.1 is an illustration of this configuration and motion. Because the vehicle possesses 2 dimensional positioning it can exhibit 3 degrees of freedom motion. This motion contains three variables of :

- Translation speed -- $v(t) \equiv \sqrt{((dx(t)/dt)^2 + (dy(t)/dt)^2)}$,

- Motion direction -- $\theta(t) \equiv \text{atan2}(dy(t)/dt, dx(t)/dt)$, if $v(t) > 0$,
- Rotational speed -- $\omega(t) \equiv d\psi(t)/dt$.

Resulting in a motion description of:

$$Q(t) \equiv (v(t), \theta(t), \omega(t)).$$

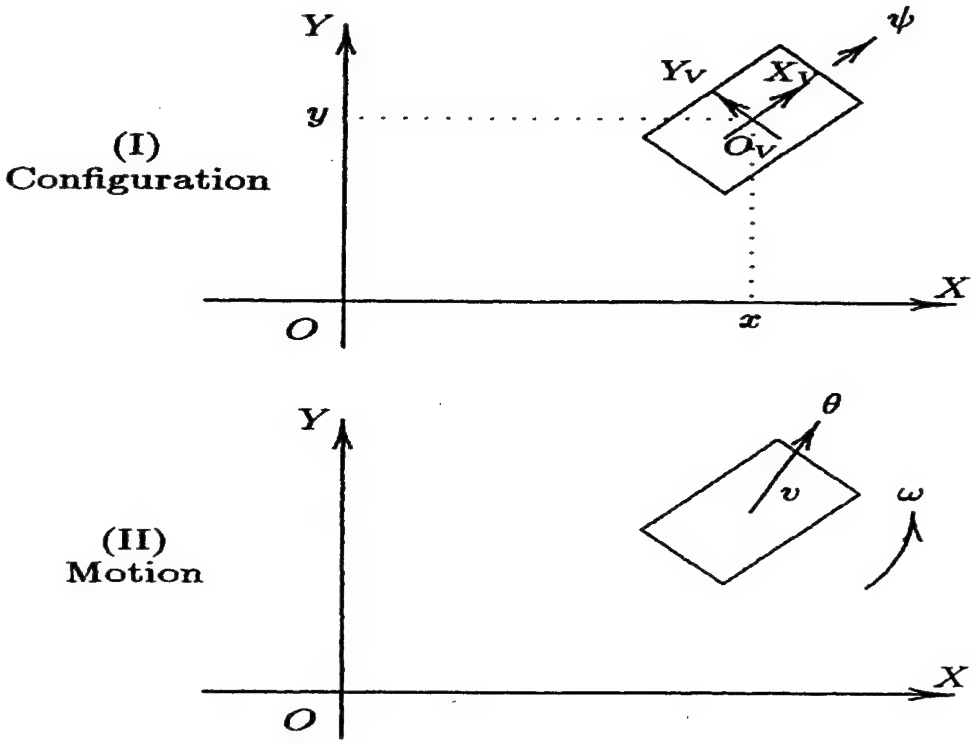

Figure 6.1: Configuration and Motion of a Rigid Body [Ref. 23].

As was mentioned in Chapter II, 3 typical vehicle motions are possible. The 3 typical motions are illustrated in Figure 6.2. Part (a) represents an all too familiar motion exhibited by a normal automobile or bicycle and is referred to as tangential motion. This is characterized by the fact that the vehicles heading orientation is equal to its translational motion direction ($d\psi(t)/dt = \theta(t)/d(t)$). Part (b) depicts a motion called constant orientation where the vehicles heading orientation is constant or rotational speed of the vehicle is 0. And part (c) shows a complex motion which incorporates rotation

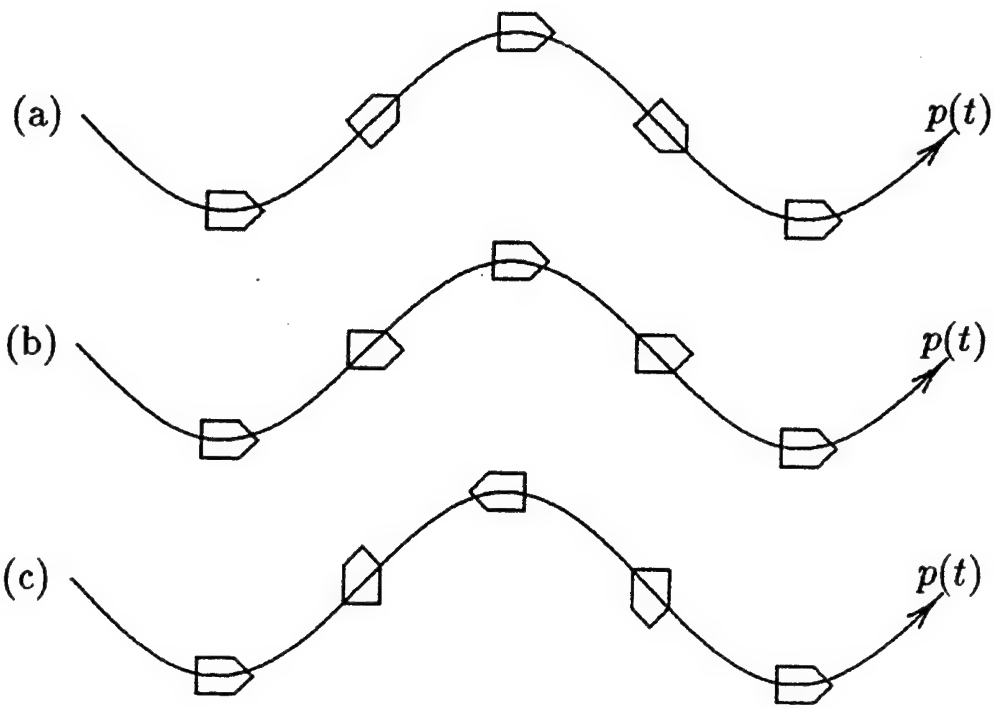

Figure 6.2: Typical vehicle motions [Ref. 23].

independently superimposed on a translational motion. This complex motion is the basis for the "Tornado" motion mode to be discussed.

B. "TORNADO" MOTION

In analyzing this motion mode, several more discriptions will have to be made clear. One is that of a point. A point is defined as

$$p_1 = (x_1, y_1) \neq (0,0)$$

described in the vehicle coordinate system. On the rotary vehicle it corresponds to a wheel . In the case of wheels 1-4, it would be (40,-40), (40,40), (-40, -40), and (-40,40). So we will have to evaluate how these wheels move while the vehicle is executing the input motion or Q. In order to evaluate this, the polar coordinate representation is

another description that is needed. This representation is defined as (ρ, α) and is represented as

$$\rho = \sqrt{(x_1^2 + y_1^2)} \text{ and } \alpha = \text{atan2}(y_1, x_1).$$

The subscript is a representation of the wheel number and can represent any wheel based on the wheel location in the vehicle coordinate system. Figure 6.3 is a representation of the composite motion of a point on a vehicle.

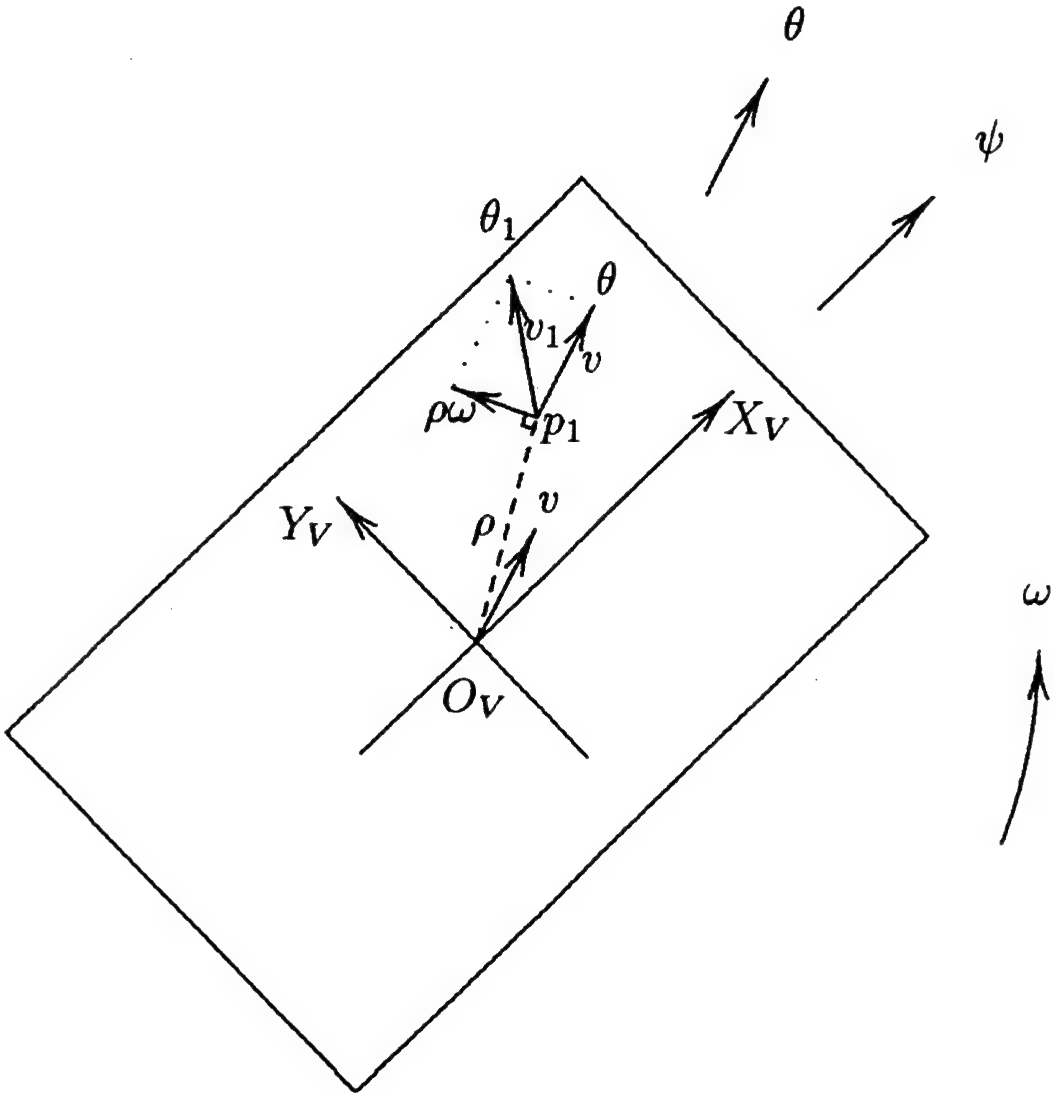

Figure 6.3: Composite Motion of a Point on a Vehicle [Ref. 23].

Given the above, a current configuration and motion of

$$q(t) = ((x(t), y(t)), \psi(t)),$$
$$Q(t) = (v(t), \theta(t), \omega(t)),$$

in the global coordinate system, the x and y-components of $v_{1x}(t)$ and $v_{1y}(t)$ in the global coordinate system can be determined mathematically [Ref. 23 p.3]. The motion speed $v_1(t)$ and direction $\theta_1(t)$ in the global coordinate system, motion direction $\theta_1{}^V(t)$ in the vehicle coordinate system, and rotation rate $\omega_1{}^V$ of p1 or any wheel is:

$$v_1(t) = \sqrt{}(\,(v_{1x}(t))^2 + (v_{1y}(t))^2)$$
$$\theta_1(t) = \mathrm{atan2}(v_{1y}(t), v_{1x}(t))$$
$$\theta_1{}^V(t) = \theta_1(t) - \psi(t)$$
$$\omega_1{}^V = ((v^2\text{⇩}+\rho^2\omega^3)+v\rho\omega(\omega+\text{⇩})\sin(\theta-\psi\ \alpha)+\rho(v\text{▫}-\text{❖}\omega)\cos((\theta-\psi-\alpha))/v_1{}^2 - \omega$$

⇩ is theta dot, ▫ is omega dot, and ❖ is motion speed dot.

Corresponding to the above equations in SRK is the following code from Appendix D (movement.c):

```
ro=whp[i].rho;
ro2=ro*ro;
beta=vehicle.heading+whp[i].alpha;
v1x = speed*cos(theta)-(whp[i].rho*omega*sin(beta));
v1y = speed*sin(theta)+(whp[i].rho*omega*cos(beta));
desiredSpeeds[i] = new_sqrt(v1x*v1x + v1y*v1y);

if (new_abs(desiredSpeeds[i]) > 0.01){
  desiredAngles[i] = atan2(v1y,v1x) - vehicle.heading;
  wheelAngleV = motion.Theta - vehicle.heading - whp[i].alpha;
  desiredAngleRates[i] = ( (speed*speed*thetaDot + ro2*Omega3)
        +speed*ro*omega*(omega+thetaDot)*sin(wheelAngleV)
        +ro*(omegaDot*speed-omega*speedDot)*cos(wheelAngleV) )
         /( desiredSpeeds[i]* desiredSpeeds[i]) - omega;
```

```
        desiredAngles0[i] = desiredAngles[i];
        desiredAngleRates0[i] = desiredAngleRates[i];
    }
```

The above code is very straight forward. A direct correlation can be discerned from the theory to the implementation. The variables were named as closely as possible to match the theory presented. The resulting code provided the mathematical computation providing the resulting values from a rotation superimposed independently on a translation motion. It is not shown, but all variables with brakets enclosing an 'i' represent a wheel. The entire routine is enclosed in a for loop which is iterated four times. Therefore, a resulting value is computed and provide to each wheel's servo for both driving and steering .

A detailed proof of the rotation rate of the moving direction at a point can be found in Ref. 24. As was stated earlier, this motion control theory is from the work of Professors Kanayama and Yun. The authors simply implemented this theory in code and applied it to the rotary vehicle.

C. JOYSTICK CONTROLLED MOTION

As discussed in chapter IV the user interface is provided by a laptop that includes a selection menu of shepherd functions. There are many options (still expanding, a work in progress) on the menu. The two of concern here are options three (3) and four (4), straight motion by joystick and XY-motion by joystick. Actually the emphasis will be on option four because option three can be considered a logical subset of option four.

The "driver" function discussed in the file movement.c is called every 10 milliseconds. One of the key functions executed under driver is the call to another function named "bodyMotion". The bodyMotion function is also located in movement.c. When a user selects option three or four from the menu a motion mode 3 or 4 is chosen. If the user chooses option four the motion mode is 4. The case the user chose causes the joystick to be read by the readJoyStick function (Appendix H, utils.c). This function reads the three ports (A, B and C) from the Intel 85C55 Parallel Port 1 (Taurus board) and converts them into an ASCII string (code segment below):

```
void readJoyStick(void)
{
 unsigned int i,index;
 unsigned char *ctrlPort=(unsigned char*)PIO1_CTRL;
 unsigned char *dataPort=(unsigned char*)PIO1_DATA;
 unsigned int pioPort1[3];
 double   a= 0.1, xx, yy, zz;

 *ctrlPort=0x9b;  /* set all ports (A,B,C) into input mode (read only) */
 index=10;        /* position for x-digits in string JOYSTICK          */

 for (i=0;i<3;i++)
  pioPort1[i] = *(dataPort+i);

 xx = (double)pioPort1[0]-128.0;
 yy = (double)pioPort1[1]-128.0;
 if (xx >= 0.0)
  xx =  xx*xx/100;
 else
  xx = -xx*xx/100;
 if (yy >= 0.0)
  yy =  yy*yy/100;
 else
  yy = -yy*yy/100;
 joyStick.x = a*(xx) + (1.0-a)*joyStick.x;
 joyStick.y = a*(yy) + (1.0-a)*joyStick.y;

 if (pioPort1[2]==0x03)
   setVME((unsigned char *)VME9210,0x00);  /* no button pressed     */
 else {
```

```
    setVME((unsigned char *)VME9210,0x02);  /* if any button pressed */
  }
}
```

It should be noted that the joystick input integer range is from [-127,128]; the intersection of the 'x' and 'y' axis on the physical joystick defines the center (x=0, y=0). Once the port is read some data smoothing is done. Due to the sensitive nature of the inputs a parabolic function was added for control (this can be seen above with the manipulation of the xx and yy variable). The purpose of the parabolic function is to ensure that when the joystick input values are small (near zero, center on the physical joystick) the slope changes will be of minimal effect, however if the input values are large (away from the physical joystick center) the effect on velocity or steering will also be proportionally large. The smoothing is continued for because of the possibility of very quick slope changes in the data being read-in. The objects joyStick.x and joyStick.y receive values that are only 10 percent (a = 0.1 in the code segment) of the xx or yy value plus 90 percent of the previous value for xx or yy. The aforementioned smoothing techniques were developed based on the experiences of Professor Kanayama an the constant "a = 0.1" determined by testing for the "best" hand feel and response.

Upon completion of the read and smoothing of the joystick data, these values (i.e., speed and theta) are passed to the wheelMotion function described in the Shepherd Motion Control Architecture (see code segment below):

```
case 4:   /*  X-Y Motion by Joystick  */
      readJoyStick();           /* ejm 19 july 97*/
      speed = -joyStick.y*0.1;    /* speed control, 0.1 determined by testing */
      theta = -joyStick.x*0.02;  /* steering control, 0.02 determined by testing */
      if (theta >  HPI)  theta =  HPI;
      if (theta < -HPI)  theta = -HPI;
      /* omega = -joyStick.omega*0.1;*/  /*  pending ejm 24 july 97 */
      break;
```

The two “if” a statements with the theta conditions above reflect the capability of the rotary vehicle to complete perpendicular driving and parking. Actually, the theta values (steering angle) of the Shepherd vehicle are unlimited, however they are constrained here for ease of use and control.

D. SEARCHING MOTION

The searching motion discussed here is based on the requirement to have a smooth technique that allows shepherd to evenly and precisely search an area for UXOs. The aforementioned search algorithm and its implementation are not trivial. The algorithm and simulation presented here are the results of a lifetime of work by Professor Kanayama. Professor Kanayama's expertise in the areas of motion planning, motion design, vehicle kinematics, sensing, guidance, learning, environmental representation, and control architectures for autonomous vehicles was the major influence. Professor Kanayama's work on the Yamabico-11 robot includes development of composite function, line tracking, circle tracking, and neutral switching technique [Ref. 7]. For the aforementioned search algorithm and simulation the composite function and line tracking technique will be used.

The goal of this simulation is to show that if given an orientation for the vehicle body and a given path, that the path can be tracked smoothly and the vehicle orientation will also change to ensure area coverage of the path traveled. Why is this important? This is critical because the desire is for the vehicle to search the path in the most safe, smooth, and efficient manner. The first assumption is that the time required to move across a path is 10 milliseconds or 0.01 seconds. Secondly, an assumption can be made for the vehicle orientation (called psi here), psi starts at 3*PI/4.0. Along the path traveled the orientation or psi will move from 3*PI/4.0 to PI/4.0 (having a net change of PI/2.0). From the aforementioned change in psi, the incremental change over time can be determined (this incremental change is called omega). Dividing the net difference in psi (Pi/2.0) derived omega by 10 milliseconds, resulting in an omega value of 0.1570796327.

A value of 40 centimeters per second was arbitrarily chosen for the vehicle velocity. For the simulation the initial vehicle body coordinates as x = 0, y = 0, and the vehicle orientation as shown above 3*PI/4.0. Also, the coordinates of the wheels must be known. Hence, we place the vehicle wheels on sides that are 80 centimeters in length (like the Shepherd vehicle).

The last item is the structure required supporting the simulation [Ref. M], below is a code segment to illustrate the aforementioned statements.

```
double deltaTime = 0.01;
double Vel = 40.0;
double omega = -0.1570796327;

typedef struct{
 double x;
 double y;  }
POINT;

typedef struct{
 POINT Point;
 double Theta;
 double Kappa;
 double Psi;
 }
CONFIGURATION;

  q_init.Point.x = 0.0;
  q_init.Point.y = 0.0;
  q_init.Theta   = 0.0;
  q_init.Kappa   = 0.0;
  q_init.Psi   = 2.356219449; /* 3*PI/4.0 */
```

```
//individual wheel coordinates
qfrontR.Point.x = 40;  /* wheel1 */
qfrontR.Point.y = -40;

qfrontL.Point.x = 40;  /* wheel2 */
qfrontL.Point.y = 40;

qrearR.Point.x = -40;  /* wheel3 */
qrearR.Point.y = -40;

qrearL.Point.x = -40;  /* wheel 4 */
qrearL.Point.y =  40;


q_xaxis.Point.x  = 0.0;  /* line to be tracked, initial configuration */
q_xaxis.Point.y  = 40.0;
q_xaxis.Theta    = 0.0;
q_xaxis.Kappa    = 0.0;
```

Another key element in the structure CONFIGURATION is theta. Theta is simply the angle to the path being tracked. For instance if the vehicle is tracking a line, when the vehicle move onto the actual line then Theta's value goes to zero. The final element required for the simulation is the step size that is used for the motion. The step size in the simulation code is called deltaS, and is vel*deltaTime (or .4 centimeters per second). Armed with this knowledge a simplified discussion can take place (for detailed knowledge of the compose function, and line tracking see Professor Kanayama's motion planning and kinematics notes [Ref. 7]).

The compose function is used to determine (using deltaS) the next position of the vehicle body. Here two compose functions are being used, one that is for the wheels (Compose2) and another for the vehicle body (Compose).

```
CONFIGURATION Compose(CONFIGURATION& q1,CONFIGURATION&
q2, CONFIGURATION& q3, double& s, double& deltaTime)
{  double x,y,
   sinTheta = sin(q1.Theta),
   cosTheta = cos(q1.Theta);

   x = q1.Point.x + q2.Point.x*cosTheta - q2.Point.y*sinTheta;
   y = q1.Point.y + q2.Point.x*sinTheta + q2.Point.y*cosTheta;
   q3.Point.x = x;
   q3.Point.y = y;
   q3.Theta = q1.Theta + q2.Theta;

   q3.Psi = q1.Psi + (omega * deltaTime);  /* how to handle move left/right? */
   fprintf(f6,"%10.3f %10.3f %10.3f %10.3f %10.3f\n",
         s,q3.Point.x, q3.Point.y,q3.Theta, q3.Psi);
   return q3;

}// end Compose
```

```
CONFIGURATION Compose2(CONFIGURATION& q1,CONFIGURATION&
q2, CONFIGURATION& q3) /*position */
{  double x,y,
   sinTheta = sin(q1.Psi),
   cosTheta = cos(q1.Psi);

   x = q1.Point.x + q2.Point.x*cosTheta - q2.Point.y*sinTheta;
   y = q1.Point.y + q2.Point.x*sinTheta + q2.Point.y*cosTheta;
```

```
q3.Point.x = x;
q3.Point.y = y;
return q3;

}// end Compose2
```

The Compose function contains several lines that are important for observations in our simulation. The calculation of q3.Theta, as mentioned before the data theta should grow in positive manner as the vehicle moves from its initial point to the line above it. Once the line that is being tracked has been reached the value for theta goes to zero. Secondly the calculation for psi shows that as the vehicle moves from the initial position to the end of the line being tracked, the value of psi will be decremented by omega*deltatime (note in the code omega is defined as a negative number [Appendix M]). The Compose2 function is important because it provides the ability to compose the body orientation (psi) with the x and y cooridinates using the previously defined step.

In the actual simulation the values of theta were manipulated to ensure the vehicle would track the next line above (40 centimeters higher) on the next step through the loop (see the code segment below):

```
q_xaxis.Point.y  = q_xaxis.Point.y + 40.0;
  if(ix%2==0){
   q_xaxis.Theta    = PI;
   q.Theta    = PI;
   omega = fabs(omega);
  }else{
   q_xaxis.Theta    = 0.0;
   q.Theta    = 0.0;
   omega = -omega;
```

The simulation proved successful based on the data provided in Appendix L. Figure 6.4 is a graphical representation of the simulation data in Appendix L. Moreover, all the structures are in place in the SRK [Appendix J, Consolidated header files] to implement the sensing motion. If more time was available for this thesis the sensing motion would have been implemented.

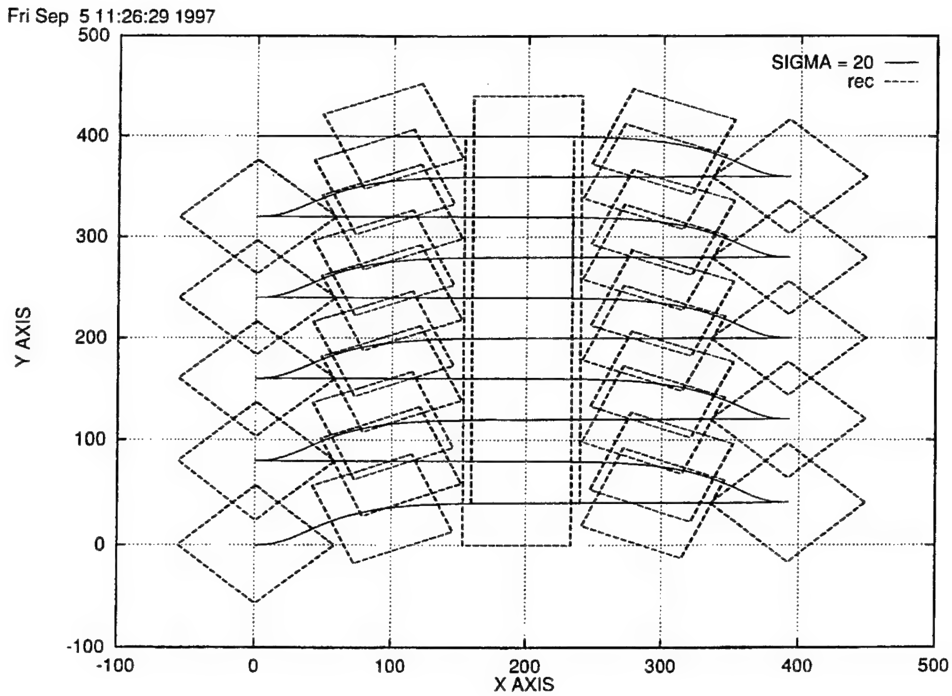

Figure 6.4: Sensing motion simulation.

VII. CONCLUSIONS

A. SUMMARY

What was accomplished as a result of this research. A restatement of the goals or questions addressed of this is necessary. The thesis was to examine the following research areas:

- What kinematics algorithms must be developed to support a vehicle with three degrees of freedom of motion? The aforementioned algorithms must support highly flexible, controlled, and precise motion.
- What types of controls are required to ensure the optimal mix of driving and steering resources? Moreover, what must be done to ensure that all the resources complement?
- How can the knowledge gained in the aforementioned research areas be used to develop searching motion?
- How should the hardware and software systems be implemented to support the aforementioned goals?

The kinematics algorithms developed to support a vehicle with 3 degrees of freedom of motion can be divided into 2 categories, 'low level' and 'high level'. The first category is 'low level'. The 'low level' algorithms are illustrated in Chapter IV and in Appendix E (motor.c). These 'low level' algorithms are concerned with taking inputs and passing the inputs to the hardware. The empirical results from the direct input and servomotor output can be seen in Chapter V. The 'high level' category considers desired body motion and its transformation into wheel motion (see Chapter IV and Appendix D). The driver function in movement.c binds the high and low level categories allowing for highly flexible, precise, and controlled motion of the rotary vehicle.

The controls required for the optimal mix of driving and steering resources are developed in Chapter V. The algorithms and their implementation as described in Chapter V removes or lessens the effect of variance and disparities of servomotor output. This ensures an optimal mix of driving and steering resources and that resources are

utilized in a complimentary manner. Hence, the aforementioned algorithms gives the desired output within 1% error for performance specifications.

The knowledge gained and the researched discribed in Chapters IV, V, and VI can be used to implement the required searching motion the rotary vehicle needs for the UXO effort. Discoveries and knowledge gained during development of the Tornado, Sinusoidal, and Joystick controlled motions; coupled with the searching motion simulation can be used to implement the tangential and searching motion (see Appendices C and D). Moreover, the structures are presently in place to support both tangential and searching motions in Appendix D.

The hardware and software systems are implemented as described in Chapters III and IV. This hardware-software implementation is tightly coupled and ensures proper communication within the system. This communcation allows the users desired inputs to be translated into vehicle motion in real-time.

B. LESSONS LEARNED

The Shepherd rotary vehicle was designed by Professor Kanayama but built by Mitsubishi Heavy Industries in Japan. This resulted in product not to specification, and a lack of legible documentation. One aspect was that the rotary vehicle could not fit into the lab it was planned for. Because, the size of the robot was not correct. Also, the documentation provided was in Japanese, big problem. None of the authors could read or speak Japanese. This generated numerous faxes and telephone calls to Mitsubishi Heavy Industries requesting support and clarification. Numerous hours were lost in this endeavor. Moreover, some of the wiring diagrams did not match the actual wiring implementation on the rotary vehicle.

The compiler used is GCC version 2.7.2.1. It was good because it was freeware. However, for cross-compiler requirements of the Motorola 68040 processor it left a lot to be desired. Most notably, the lack of all basic libraries. All input-output functions and math functions had to be written by the authors. Again, taking an inordinate amount of time and effort. Here the authors were recreating the wheel, per se. Also, the unusual

handling of structure passing required the development of several work-arounds. Essentially, standard C-code writing had to be modified extensively. The Motorola 68040 chip was designed to support code written for the Motorola 68020. This may been true but the authors determined that for math functions this was not the case. This also may be related to the compiler (switches). The authors contacted Motorola and Omnibyte Corporations for help in dealing with these matters with little help provided. In the majority of cases the help provided led to dead-ends. Technicians at Omnibyte recommended that a purchase of a compiler suitable for the M68040 be made.

The Taurus Bug (firmware on the Taurus Board) provided excellent capabilities. However, a lot of time was required to use the tools. On many occasions numerous hours were spent looking through loops of assembly code. This would not have been the case with a modern debugger.

Group cross-pollenation is essential for a project with multiple disciplines involved. For example, a low voltage problem on the rotary vehicle caused the CPU to slow or halt on some occasions. Many hours were spent trying to determine what was wrong with the C-coding and the problem was hardware. This may have been alleviated had there been an electrical engineer on the team. Problems like this were exacerbated by the documentation problem mentioned earlier. Another problem was the overheating of the boards on the VMEbus (including the CPU) resulting in system shutdowns as well as aberrations in CPU behavior.

C. RECOMMENDATIONS FOR FUTURE RESEARCH

Recommendations for future research should include:

- Implementation of tangential motion, the rudiments of which are already in place.
- Implementation of sensing motion which is required for the percise, accurate, and safe detection of UXO's.
- Transitioning the SRK to the PC environment possibly using LINUX freeware as the real-time operating kernal.

- Implementation of a M68040 specific compiler with all libraries.
- Addition of the robot arm and implementation of the a neural net and expert system for UXO identification. Considerations should be given to sensors such as a magnetometer, digital camera devices, ground penetrating radar, xray devices and GPS.
- Implementation of wireless ethernet or hand held computing devices for control and monitoring of the rotary vehicle by a user who is not co-located.
- Possible missions other than UXO. For instance, an unmanned scout vehicle or mobile sentry.

It is obvious from the above recommendations that the possible uses of the rotary vehicle is extensive and varied. The highly flexible, precise, and controlled motion of the vehicle makes it an ideal platform for many ground applications.

APPENDIX A: SOURCE CODE (MAKEFILE)

The following code was modified by: Professor Kanayama, Thorsten Leonardy, Edward Mays, and Ferdinand A. Reid.

```
# -------------------------------------------------------------------------- *
#                                                                             *
# File:       M A K E F I L E                                                 *
#                                                                             *
# Environment:  GCC Compiler v2.7.2                                           *
# Update:     02 February 1997 (Leonardy)                                     *
#             02 April 1997    (Ed Mays)                                      *
#             14 May 1997      (Leonardy, added utils030.*)                   *
# Name:       Thorsten Leonardy                                               *
# Purpose:    Makefile for project S H E P H E R D .                          *
#                                                                             *
# Invoke:     make comp (to generate code)                                    *
#             make print (to print all files to printer ap1 in Sp-511)       *
#             make clean (to clean directory from object files)               *
#                                                                             *
# --------------------------------------------------------------------------*/


comp: startup.o shepherd.o timer.o serial.o math.o utils.o utils030.o user.o \
        motor.o movement.o
        ld  -Ttext 0x10000 -Tdata 0x20000 -Tbss 0x30000 -Map shepherd.map -oformat
        srec \
```

```
	-o shepherd.TXT startup.o shepherd.o timer.o serial.o math.o utils.o \
	utils030.o user.o motor.o movement.o

shepherd.o : shepherd.c
	gcc -c -m68040 -o shepherd.o shepherd.c

timer.o : timer.c
	gcc -c -m68040 -o timer.o timer.c

serial.o : serial.c
	gcc -c -m68040 -o serial.o serial.c

#servo.o : servo.c
#	gcc -c -m68040 -o servo.o servo.c

utils.o : utils.c
	gcc -c -m68040 -o utils.o utils.c

utils030.o : utils030.c
	gcc -c -m68040 -o utils030.o utils030.c

user.o : user.c
	gcc -c -m68040 -o user.o user.c

motor.o :  motor.c
	gcc -c -m68040 -o  motor.o  motor.c
```

```
movement.o : movement.c
	gcc -c -m68040 -o movement.o movement.c
startup.o :  startup.s
	as -o startup.o startup.s
math.o: math.c
	gcc -c -m68040 -o math.o math.c
# This cleans out everything except the Makefile,
# and source files
clean:; rm -f *.o core
# This prints all source files to the printer ap1 in Sp-511
print:; enscript -2r -g -Pap1 makefile shepherd.map shepherd.h shepherd.c \
	user.h user.c utils.h utils.c utils030.c serial.h serial.c servo.h servo.c \
	timer.h timer.c movement.h movement.c wheeldrive.h wheeldrive.c \
	math.h math.c startup.s
# This prints all source files to the printer sp1 in Sp-527
prsp1:; enscript -2r -g -Psp1 makefile shepherd.map shepherd.h shepherd.c \
	user.h user.c utils.h utils.c utils030.c serial.h serial.c servo.h servo.c \
	timer.h timer.c movement.h movement.c navigat.h navigat.c  wheeldrive.h
```

```
        wheeldrive.c \
        math.h math.c motor.h motor.c startup.s

#**********************************
# End of makefile
# *********************************\
```

APPENDIX B: SOURCE CODE (SHEPHERD.C)

The following code was modified by: Professor Kanayama, Thorsten Leonardy, Edward Mays, and Ferdinand A. Reid.

```
/* ---------------------------------------------------------------------- *
 *                                                                        *
 * File:       S H E P H E R D . C                                        *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Last update:  29 January 1997                                          *
 * Name:         Thorsten Leonardy                                        *
 * Purpose:      Provides the kernel for SHEPHERD.                        *
 *                                                                        *
 * Compiled:     >gcc -c -m68040 -o shepherd.o shepherd.c                 *
 *                                                                        *
 * ---------------------------------------------------------------------- */

#include "shepherd.h" /* general defines       */
#include "movement.h"


/* -------------------------- *
 * constant character strings *
 * -------------------------- */
unsigned char JOYSTICK[26]= {
```

```
        27,91,49,56,59,54,48,72,        /* ESC [ '1' '8' ; '6' '0' H */
        120,61,48,48,48,32,            /* 'x=000 '          */
        121,61,48,48,48,32,            /* 'y=000 '          */
        122,61,48,48,48,0};            /* 'z=000'           */

unsigned int wheelEncoder[8];
unsigned char bcdString[25]=
        {24,27,91,48,50,59,52,48,72,
        48,48,48,48,48,48,48,48,48,48,48,48,48,48,48,48};

/* clock contains the current date/time updated every second in ascii format */
/* it starts with a count byte, the cursor positioning sequence followed by */
/* the date-group and the time group in the form:                          */
/*   clock=".........mm-dd-yy hh:mm:ss." where '.' is part of count or ESC */
/* sequence.                                  */

/*unsigned char clock[27]={ */
/*          25,27,91,48,49,59,54,52,72,  */     /* ESC [ '0' '1' ; '6' '4' H */
/*       109,109,45,100,100,45,121,121,32, */  /* 'mm-dd-yy '          */
/*        104,104,58,109,109,58,115,115,0};*/  /* 'hh:mm:ss'           */

void advanceCount()
{
 edCounter++;
}

```

```
main()
{
   initBoards();      /* initialize boards                          */
   initMovement();   /* initialize movement                        */
   sioInit();        /* initialize 68C681 DUART for serial I/O          */
   timerStart();     /* initialize and start timer-5 fopr motion control  */
   /* setup68030();*/ /* setup OMNI Module 0 for serial I/O to VT100      */
   enable();         /* enable all interrupts, user mode            */
   user();           /* let user handle the main portion            */
   disable();        /* disable interrupts, supervisor mode         */

   /* here goes downloading stuff for analysis ... (i.e. copy memory      */
   /* from Taurus Main memory to host computer (Laptop or SparcStation)     */

   return;

} /* end of main() */



/*********************************************************************
 End of shepherd.c
 *********************************************************************/



```

APPENDIX C: SOURCE CODE (USER.C)

The following code was modified by: Professor Kanayama, Thorsten Leonardy, Edward Mays, and Ferdinand A. Reid.

```
/* ------------------------------------------------------------------ *
 *                                                                    *
 * File:        U S E R . C                                           *
 *                                                                    *
 * Environment:  GCC Compiler v2.7.2                                  *
 * Last update:  18 February 1997                                     *
 * Name:         Thorsten Leonardy                                    *
 * Purpose:      Provides the userpart for SHEPHERD.                  *
 *                                                                    *
 * Compiled:     >gcc -c -m68040 -o user.o user.c                     *
 *                                                                    *
 * ------------------------------------------------------------------ */

#include "shepherd.h" /* general defines        */


/* -------------------------------- *
 * Global variables for test program *
 * -------------------------------- */

extern unsigned char inPortA; /* defined in serial.c */
extern unsigned char vt100xy[9]; /* ESC-Sequence for Cursor Position */
```

```
extern unsigned char clrSCR[5];  /* ESC-Sequence for clear Screen    */
extern unsigned char clrLine[6]; /* ESC-Sequence for clear line      */
extern unsigned char prtSCR[4];  /* ESC-Sequence for print screen    */
extern unsigned char cursorOFF[5];/* ESC-Sequence for cursor off     */

extern void gotoXY(int,int);
extern int wheelEncoder[8];  /* defined in shepherd.c */


extern char bcdString[]; /* defined in shepherd.c */

unsigned short velocity=0;


/* -------------------------- *
 * constant character strings *
 * -------------------------- */

#define MENU_LINES 24
char *menu[MENU_LINES]={
   "SHEPHERD Main Menu (Last Update: 27 Feb 97)\n\r",
   "1---5----0----5----0----5----0----5----0----5----0----5----0----5----0----5----0\n\r",
   "\n\r"
   "Please choose:            Diagnostics\n\r",
   "                          ------------------------------------------------\n\r",
    /* character pressed, analize ... */
```

```
"(1) Stop                        \n\r",
"(2) Straight Motion (Autonomous)   \n\r",
"(3) Straight Motion by Joystick    \n\r",
"(4) XY-Motion by Joystick          \n\r",
"(5) Rotate                         \n\r",
"(6) Sinusoidal Motion              \n\r",
"(7) Tornado (External Center of Rotation)\n\r",
"(8) Tornado (Internal Center of Rotation)\n\r",
"(9) Tangential Motion              \n\r",
"(0) Exit Program                   \n\r",
"(a) Tangential Motion(II)          \n\r",
"(t) Test wheel Angle 4             \n\r",
" \n\r",
" \n\r",
" \n\r",
" \n\r",
" \n\r",
" \n\r",
"-----------------------------------------------------------------------------\n\r",
" "
};
```

```
/* ---------------------------------------------------------------------- *
 * displayMenu()                                                          *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Last update:  27 January 1997                                          *
 * Name:         Thorsten Leonardy                                        *
 * Purpose:      This function outputs a menu to the screen              *
 *                                                                        *
 * ---------------------------------------------------------------------- */
void displayMenu(void)
{
  int i;
  sioOut(0,clrSCR);   /* clear screen */
  gotoXY(1,1);
  for (i=0;i<MENU_LINES; i++)
     sioOut(0,menu[i]);
  return;
}
/* ---------------------------------------------------------------------- *
 * status()                                                               *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Last update:  18 February 1997                                         *
```

```
 * Name:        Thorsten Leonardy                              *
 * Purpose:     This function outputs a status line at the bottom of the    *
 *              screen.                                        *
 * ------------------------------------------------------------------- */
void status(unsigned char *p)
{
  gotoXY(24,1);       /* position cursor */
  sioOut(0,clrLine);  /* clear line      */
  sioOut(0,p);        /* print text      */
  return;
}


/* ------------------------------------------------------------------- *
 * convertBCD()                                                        *
 *                                                                     *
 * Environment:  GCC Compiler v2.7.2                                   *
 * Last update:  20 February 1997                                      *
 * Name:        Thorsten Leonardy                                      *
 * Purpose:     This function converts an unsigned integer to a BCD string  *
 *              of 16 characters. The value is right justified with leading *
 *              zeros.                                                 *
 * ------------------------------------------------------------------- */
void convertBCD(unsigned char *s, unsigned int data)
{
  int i=15;
```

```

 for (i=15; i>=0; i--) {  /* write 16 Bytes, adjust integer to right of string */
   *(s+i)=48+(data%10);
   data=data/10;
 }
 return;
}

/* ----------------------------------------------------------------------- *
 * convertInt()                                                             *
 *                                                                          *
 * Environment:  GCC Compiler v2.7.2                                        *
 * Last update:  11 June  1997                                              *
 * Name:         Thorsten Leonardy, Yutaka Kanayama                         *
 * Purpose:      This function converts a signed integer to a BCD string    *
 *               of 16 characters. The value is right justified with leading *
 *               zeros.                                                     *
 * ----------------------------------------------------------------------- */
void convertInt(unsigned char *s, int data)
{
 int i=15;

 if (data >= 0)
   *s = 32;  /*  space     for positive number */
 else
   {
```

```
  *s = 45;  /*  minus sign for negative number */
  data = -data;
 }

for (i=15; i>=1; i--) {  /* write 16 Bytes, adjust integer to right of string */
  *(s+i)=48+(data%10);
  data=data/10;
 }
return;
}
/* ------------------------------------------------------------------------ *
 * wheelDrive()                                                              *
 *                                                                           *
 * Environment:  GCC Compiler v2.7.2                                         *
 * Last update:  27 February 1997                                            *
 * Name:         Thorsten Leonardy                                           *
 * Purpose:      This function drives the specified wheel with required      *
 *               velocity.                                                   *
 *                                                                           *
 * ------------------------------------------------------------------------ */
void wheelDrive1(unsigned short wheel, unsigned short velo)
{
 unsigned short *servoOut=(unsigned short *)0xffff0482;  /* analog out */
```

```
unsigned int   *servoControl=(unsigned int*)0xffffff00;   /* Data out   */

*servoOut=velo;       /* set velocity first */

if (wheel)
  *servoControl=0x00000004;
else
  *servoControl=0x00000000;

return;
}





/* ---------------------------------------------------------------------- *
 * updateWheelStatus()                                                     *
 *                                                                         *
 * Environment:  GCC Compiler v2.7.2                                       *
 * Last update:  27 February 1997                                          *
 * Name:        Thorsten Leonardy                                          *
 * Purpose:     This function reads the current shaft encoder readings for *
 *              all eight servo motors and outputs them to the screen.     *
 *                                                                         *
 * unsigned int wheelEncoder[8]  - array to hold the shaft encoder readings *
```

```
 * unsigned char *bcdString      - string to hold converted encoder reading  *
 *                                                                           *
 * ------------------------------------------------------------------------ */
void updateWheelStatus(void)
{

  unsigned short i,posx,posy;

  readWheelStatus(wheelEncoder);        /* read wheel status: File servo.c */

  posx=8;                     /* x-position on screen for reading motor 1 */
  posy=40;                    /* y-position on screen for reading motor 1 */

  for (i=0; i<8; i++) {
    posx=8+i%4;                         /* position for x        */
    posy=40+20*(i/4);                   /* position for y        */
    bcdString[3]=48+posx/10;            /* convert tens to ascii  */
    bcdString[4]=48+posx%10;            /* convert ones to ascii  */
    bcdString[6]=48+posy/10;            /* convert tens to ascii  */
    bcdString[7]=48+posy%10;            /* convert ones to ascii  */
    convertBCD(bcdString+9,wheelEncoder[i]);   /* convert reading itself */
    WRITE_ENCODER();                    /* output ascii          */
  } /* end of for */

  return;
}
```

```
void displayDriveSpeed()
{
 double speed00,speed0,speed1,speed2,speed3;

 disable();
 speed00=desiredSpeeds[0];
 speed0=Display_Speeds[0];
 speed1=Display_Speeds[1];
 speed2=Display_Speeds[2];
 speed3=Display_Speeds[3];
 Display_Speeds[0]=0.0;
 Display_Speeds[1]=0.0;
 Display_Speeds[2]=0.0;
 Display_Speeds[3]=0.0;

 enable();

    convertInt(bcdString+9,speed00);
     bcdString[3]='0';
     bcdString[4]='3';
     bcdString[6]='4';
     bcdString[7]='0';
     sioOut(0,bcdString);
    convertInt(bcdString+9,speed0);
```

```
    bcdString[3]='0';
    bcdString[4]='3';
    bcdString[6]='6';
    bcdString[7]='0';
    sioOut(0,bcdString);
  convertInt(bcdString+9,speed1);
    bcdString[3]='0';
    bcdString[4]='4';
    bcdString[6]='6';
    bcdString[7]='0';
    sioOut(0,bcdString);
  convertInt(bcdString+9,speed2);
    bcdString[3]='0';
    bcdString[4]='5';
    bcdString[6]='6';
    bcdString[7]='0';
    sioOut(0,bcdString);
  convertInt(bcdString+9,speed3);
    bcdString[3]='0';
    bcdString[4]='6';
    bcdString[6]='6';
    bcdString[7]='0';
    sioOut(0,bcdString);
return;
}
```

```
void displayDriveSteer()
{
 double steer00,steer0,steer1,steer2,steer3;

 disable();
 steer00=Steer_Digits[0];
 /* steer00=desiredAngleRates[0]; */
 steer0=desiredAngleRates[0]*1000;
 steer0=Display_Steers[0];
 steer1=Display_Steers[1];
 steer2=Display_Steers[2];
 steer3=Display_Steers[3];
 Display_Steers[0]=0.0;
 Display_Steers[1]=0.0;
 Display_Steers[2]=0.0;
 Display_Steers[3]=0.0;

 enable();

    convertInt(bcdString+9,steer00);
     bcdString[3]='0';
     bcdString[4]='3';
     bcdString[6]='4';
     bcdString[7]='0';
     sioOut(0,bcdString);
    convertInt(bcdString+9,steer0);
```

```
     bcdString[3]='0';
     bcdString[4]='3';
     bcdString[6]='6';
     bcdString[7]='0';
     sioOut(0,bcdString);
  convertInt(bcdString+9,steer1);
     bcdString[3]='0';
     bcdString[4]='4';
     bcdString[6]='6';
     bcdString[7]='0';
     sioOut(0,bcdString);
  convertInt(bcdString+9,steer2);
     bcdString[3]='0';
     bcdString[4]='5';
     bcdString[6]='6';
     bcdString[7]='0';
     sioOut(0,bcdString);
  convertInt(bcdString+9,steer3);
     bcdString[3]='0';
     bcdString[4]='6';
     bcdString[6]='6';
     bcdString[7]='0';
     sioOut(0,bcdString);
return;
}

```

```
void displayAngles()
{
 double steer0,steer1,steer2,steer3;
 disable();
 steer0=actualAngles[0]*1000*RadsToDegrees;
 steer1=actualAngles[1]*1000*RadsToDegrees;
 steer2=actualAngles[2]*1000*RadsToDegrees;
 steer3=actualAngles[3]*1000*RadsToDegrees;
 enable();
   convertInt(bcdString+9,steer0);
     bcdString[3]='0';
     bcdString[4]='3';
     bcdString[6]='6';
     bcdString[7]='0';
     sioOut(0,bcdString);
   convertInt(bcdString+9,steer1);
     bcdString[3]='0';
     bcdString[4]='4';
     bcdString[6]='6';
     bcdString[7]='0';
     sioOut(0,bcdString);
   convertInt(bcdString+9,steer2);
```

```
    bcdString[3]='0';
    bcdString[4]='5';
    bcdString[6]='6';
    bcdString[7]='0';
    sioOut(0,bcdString);
  convertInt(bcdString+9,steer3);
    bcdString[3]='0';
    bcdString[4]='6';
    bcdString[6]='6';
    bcdString[7]='0';
    sioOut(0,bcdString);
return;
}
void displayVehicleConfig()
{
 double coordx, coordy, heading, kappa;

 disable();

 coordx  = vehicle.coord.x;
 coordy  = vehicle.coord.y;
 heading = vehicle.heading;
 kappa   = vehicle.kappa;

 enable();

```

```
convertInt(bcdString+9,coordx);
  bcdString[3]='0';
  bcdString[4]='3';
  bcdString[6]='6';
  bcdString[7]='0';
  sioOut(0,bcdString);
convertInt(bcdString+9,coordy);
  bcdString[3]='0';
  bcdString[4]='4';
  bcdString[6]='6';
  bcdString[7]='0';
  sioOut(0,bcdString);
convertInt(bcdString+9,heading);
  bcdString[3]='0';
  bcdString[4]='5';
  bcdString[6]='6';
  bcdString[7]='0';
  sioOut(0,bcdString);
convertInt(bcdString+9,kappa);
  bcdString[3]='0';
  bcdString[4]='6';
  bcdString[6]='6';
  bcdString[7]='0';
  sioOut(0,bcdString);
return;
}
```

```
/* ---------------------------------------------------------------------- *
 * user()                                                  *
 *                                                         *
 * Environment:  GCC Compiler v2.7.2                                  *
 * Last update:  18 February 1997                               *
 * Name:         Thorsten Leonardy                              *
 * Purpose:      This function provides the user shell.              *
 *                                                         *
 * ---------------------------------------------------------------------- */

void user(void)
{
 int a;
 char *s;
 unsigned int   *servoControl=(unsigned int *)VME2170;/* test only */
  displayMenu();  /*    display menu      */
 do
  {
    inPortA='?';            /* reset character                 */
    while(inPortA=='?');      /* wait for character to be typed in */
    /* character pressed, analize ... */
    switch(inPortA)
        {
      case '1' : if (mode != 5)     /* Stop */
                    {
                      mode0state = 0;
```

```
                mode = 1;
                while (mode0state ==0) {};
                disable();
                align();
                enable();





                /*  *servoControl=0x00429429; test by Ed */
                mode0state = 2;
              }
          else
            {
                mode = 1;
                disable();
                alignAfterRotate();
                enable();
                /* *servoControl=0x00429429;   test by Ed*/

              }
          initMovement();
          break;

case '2' : mode = 2;  /* Straight Motion (Autonomous)  */
```

```
        break;

    case '3' : mode = 3;  /* Straight Motion by Joystick  */
        break;

    case '4' : mode = 4;  /* X-Y Motion by Joystick */
        /*  for (a=0;a<100;a++){                         */
        /*     while ((edCounter % 200 != 0) && (a != 100)){*/
        /*        displayAngles();                        */
        /*     }                                          */
        /*  }                                             */
        break;

    case '5' : mode5state = 0;  /* Rotate       */
           mode = 5;
        break;

    case '6' : mode = 6;  /* Sinusoidal Motion  */
        break;

    case '7' : mode = 7;  /* Tornado (Center of Rot External)    */
        break;

    case '8' : mode = 8;  /* Tornado (Center of Rot Internal)    */
        break;

```

```
case '9' : mode = 9;  /* Tangential Motion */
                initTangent();
                while(1){
                        while(edCounter%200 != 0){};
                        displayVehicleConfig();
                };
                break;

    case 'a' : mode = 10; /* Tangential Motion (II) */
            break;

    case 't' : modeTstate = 0;  /* Steering test mode */
          /*  Flag = 1;  initialized in movement.c */
            mode = 100;
               while (1)
                {
                  oldFlag = Flag;
                  while (Flag == oldFlag) {}
                  displayAngles();
                }
            break;

  default  :  break;
  } /* end of switch */
} while(inPortA!='0');/* end of while, exit with '0' entered at keyboard */
```

```
sioOut(0,clrSCR);      /* clear screen */
sioOut(0,"\n\r\n\r");  /* some cr,lf   */

return;

while(1)
  {
   while(edCounter%200 != 0){};
   /* displayJoyStick();   */
   displayDriveSteer();
   /* displayAngles();    */
  };
sioOut(0,cursorOFF); /* switch cursor off (no blink) */

} /* end of user() */



asm("
     .even
     .text
     .globl _WRITE_ENCODER

_WRITE_ENCODER:

     pea.l _bcdString
```

```
    trap #15
    dc.w 0x0023

    rts
");



/****************************************************************
End of user.c
****************************************************************/

```

APPENDIX D: SOURCE CODE (MOVEMENT.C)

The following code was modified by: Professor Kanayama, Thorsten Leonardy, Edward Mays, and Ferdinand A. Reid.

```
#include "shepherd.h"
#include "movement.h"
#include "math.h"

/* ------------------------------------------------------------ *
 * Main                                                         *
 * ------------------------------------------------------------ */
void driver()
{
  readEncoders();  /*  Read Drive/Steer Motors   */
  computeActualRates();
  /*accumulateDriveSpeed();  only for wheel speed displaying */
  accumulateDriveSteer();
  bodyMotion();
  wheelMotion();
  /*  testDrive1();  */
  driveMotors();
  advanceCount();
}
```

```
/* ------------------------------------------------------------ *
 * Initialize Movement:                                          *
 * intialize Configuration and vehicle motion                    *
 * ------------------------------------------------------------ */
void initMovement()
{
  int ix;

  Flag = 1;
  oldMode = 0;
  mode = 1;
  Omega_Speed=0.0;
  testCounter=0;
  edCounter=0;
  pathLength=0.0;

  K1[0]=11.448;/*0<=speed<=5,*/
  K1[1]=11.500;
  K1[2]=11.496;
  K1[3]=12.375;
```

```
K2[0]=11.500;/*5>speed<8*/
K2[1]=11.500;
K2[2]=11.644;
K2[3]=12.000;


K3[0]=11.611;/*8>=speed<20*/
K3[1]=11.585;
K3[2]=11.686;
K3[3]=11.840;

K4[0]=11.711;/*20>=speed<=70*/
K4[1]=11.659;
K4[2]=11.705;
K4[3]=11.727;

K6[0]=11.710;/*70>speed<K5*/
K6[1]=11.700;
K6[2]=11.700;
K6[3]=11.715;

```

```
DigitToCmDrive[0]= +0.0011369287;/* driving constant cm/count = digitToRadDrive*18.9cm*/
DigitToCmDrive[1]= -0.0011369287;
DigitToCmDrive[2]= +0.0011369287;
DigitToCmDrive[3]= -0.0011369287;


motion.Speed=0.0;
motion.Theta=0.0;
motion.Omega=0.0;

radius = 100;

vehicle.coord.x=0.0;
vehicle.coord.y=0.0;
vehicle.heading=0.0;
vehicle.kappa=1/radius;

ai[0] = 40.0; ai[1] = 40.0; ai[2] =-40.0; ai[3] =-40.0;
bi[0] =-40.0; bi[1] = 40.0; bi[2] =-40.0; bi[3] = 40.0;
```

```
joyStick.x = 0.0;
joyStick.y = 0.0;

setupPolar(whp);

for (ix =0; ix <ARRAY_SIZE; ix++){
  PreviousCountSpeed[ix]=99999999;
  PreviousCountSteer[ix]=99999999;
  Display_Speeds[ix]=0.0;
  Display_Steers[ix]=0.0;
  actualAngles[ix]=0.0;
  desiredSpeeds[ix] = 0.0;
  desiredAngleRates[ix] = 0.0;
  desiredAngleRates0[ix] = 0.0;
  desiredAngles[ix]=0.0;
  desiredAngles0[ix]=0.0;
  WheelDirAct0[ix]= 1.0e8;
  WheelDirAct[ix] = 0.0;
  WheelDirDes[ix] = 0.0;
  steerReadings[ix]=0.0; /* not used only testing */
  driveReadings[ix]=0;
```

```
  }
}


/* ---------------------------------------------------------------- *
 * SetupPolar                                         *
 * ---------------------------------------------------------------- */
void  setupPolar(polar  whp[4])
{
  whp[0].rho = whp[1].rho = whp[2].rho = whp[3].rho = 56.5685425;
          /* distances = 40 * sqrt(2) */
  whp[0].alpha = -QPI;          /*  front right wheel 1   */
  whp[1].alpha =  QPI;          /*  front left  wheel 2   */
  whp[2].alpha = -3.0*QPI;      /*  rear  right wheel 3   */
  whp[3].alpha =  3.0*QPI;      /*  rear  left  wheel 4   */
}


/* ---------------------------------------------------------------- *
 * bodyMotion -- Updates Vehicle                             *
 * ---------------------------------------------------------------- */
void bodyMotion()
```

```
{
double v0, omega0,
     linSpeed= 4.0,
     linAcc  = 1.0,
     rotSpeed= 0.1,          /* 0.05,       */
     rotAcc  = 0.025,        /* 0.0125;     */
     RPI     = QPI*1.5;      /* 67.5 degrees */
double theta, omega, speed;

speed = motion0.Speed = motion.Speed;  /* save the previous motion   */
theta = motion0.Theta = motion.Theta;  /* for computing derivatives  */
omega = motion0.Omega = motion.Omega;

switch(mode){
  case 1:
    if (mode0state == 2)
    break;
    if ( (Speed_Digits[0] == 0) && (Speed_Digits[1] == 0) &&
        (Speed_Digits[2] == 0) && (Speed_Digits[3] == 0) &&
        (Steer_Digits[0] == 0) && (Steer_Digits[1] == 0) &&
        (Steer_Digits[2] == 0) && (Steer_Digits[3] == 0))
      mode0state = 1;
```

```
          /* allStop();   will be inserted later */
   break;

 case 2:
   speed = min(speed + 2.0*DeltaT, 10.0);
   break;

 case 3:                         /* Straight Motion by Joystick  */
   readJoyStick();               /* ejm 19 july 97            */
   speed = -joyStick.y*0.1;
   theta = 0.0;
   omega = 0.0;
   break;

 case 4:   /*  X-Y Motion by Joystick  */
   readJoyStick();            /* ejm 19 july 97*/
   speed = -joyStick.y*0.1;    /* speed control  */
   theta = -joyStick.x*0.02;  /* steering control */
   if (theta >  HPI)  theta =  HPI;
   if (theta < -HPI)  theta = -HPI;
   /* omega = -joyStick.omega*0.1;*/  /*   24 july 97 */
   break;
```

```
case 5:
  if (mode5state == 1){
       readJoyStick();
       speed = -joyStick.y*0.1;
  }
  break;

case 6:  /* sinusoidal motion  */
  speed = min(speed + linAcc*DeltaT, 10.0);
  speed = speed;
  if (speed == 10.0){
    pathLength += DeltaT*speed;
    theta = 0.4 * sin(pathLength/20.0);  /* sine curve motion  */
  }
  break;

case 7:          /*  Tornado External   */
  speed = min(speed + 1.0*DeltaT, 8.0);
  if ( speed == 8.0)
    omega = min(omega + 0.0125*DeltaT, 0.1);   /* radius = 80 cm */
  break;
```

```
  case 8:         /*  Tornado Internal   */
    speed = min(speed + 1.0*DeltaT, 8.0);
    if ( speed == 8.0)
      omega = min(omega + 0.025*DeltaT, 0.2);     /* radius = 40 cm */
    break;

  case 9:  /* tangential motion */
    tangentialMotion();
    break;

  case 10:  /* tangential motion (II) */
    speed = min(speed + linAcc*DeltaT, 8.0);
    break;

  case 100:
    break;
}

if (mode != 9){
  motion.Speed = speed;
  motion.Theta = theta;
```

```
    motion.Omega = omega;

    vehicle.heading = vehicle.heading + motion.Omega*DeltaT;
    vehicle.coord.x = vehicle.coord.x + motion.Speed*DeltaT * cos(motion.Theta);
    vehicle.coord.y = vehicle.coord.y + motion.Speed*DeltaT * sin(motion.Theta);

    speedDot=(motion.Speed - motion0.Speed)/DeltaT;
    thetaDot=(motion.Theta - motion0.Theta)/DeltaT;
    omegaDot=(motion.Omega - motion0.Omega)/DeltaT;
  }
}


/* ------------------------------------------------------------ *
 * wheelMotion                                                   *
 * ------------------------------------------------------------ */
void wheelMotion()
{     /*the function that truly belongs here is in calculate.org */
  int i;
  double v1x, v1y, v1yv1xRatio;
  double theta=motion.Theta,
         omega=motion.Omega,
```

```
     speed=motion.Speed,
     Omega2=omega*omega,
     Omega3=Omega2*omega,
     beta,ro,ro2,
     wheelAngleV;

if (mode == 5){          /* rotate case  */
  switch(mode5state){
      case 0:
        /* turn each wheel by +-PI/4 in 5 seconds  */
        desiredAngles[0] += QPIby500;  /* = (PI/4)/500  */
        desiredAngles[1] -= QPIby500;
        desiredAngles[2] -= QPIby500;
        desiredAngles[3] += QPIby500;
        if (desiredAngles[0] >= QPI)
          mode5state = 1;
          break;

      case 1:           /*  drive wheels to rotate body  */
        desiredSpeeds[0] = +speed;
        desiredSpeeds[1] = -speed;
        desiredSpeeds[2] = +speed;
```

```
          desiredSpeeds[3] = -speed;
          break;
    }
    return;
  }

for (i=0; i < 4; i++){     /* non-rotate case */
  ro=whp[i].rho;
  ro2=ro*ro;
  beta=vehicle.heading+whp[i].alpha;
  v1x = speed*cos(theta)-(whp[i].rho*omega*sin(beta));
  v1y = speed*sin(theta)+(whp[i].rho*omega*cos(beta));
  desiredSpeeds[i] = new_sqrt(v1x*v1x + v1y*v1y);

  switch(mode){
      case 1:
      case 2:
      case 3:
        if (speed < 0.0)
          desiredSpeeds[i] = -desiredSpeeds[i];
        if (new_abs(v1x) > 0.01){
          v1yv1xRatio=v1y/v1x;
```

```
        desiredAngles[i] = atan(v1yv1xRatio) - vehicle.heading;
        wheelAngleV = motion.Theta - vehicle.heading - whp[i].alpha;
        desiredAngleRates[i] =
            ( (speed*speed*thetaDot + ro2*Omega3)
              +speed*ro*omega*(omega+thetaDot)*sin(wheelAngleV)
              +ro*(omegaDot*speed-omega*speedDot)*cos(wheelAngleV) )
            /( desiredSpeeds[i]* desiredSpeeds[i]) - omega;
        desiredAngles0[i] = desiredAngles[i];
        desiredAngleRates0[i] = desiredAngleRates[i];
      }
      else{
        desiredAngles[i] = desiredAngles0[i];
        desiredAngleRates[i] = desiredAngleRates0[i];
      }
      break;

  case 4:
     if (speed < 0.0)
       desiredSpeeds[i] = -desiredSpeeds[i];
     if (new_abs(v1x) > 0.01){
       v1yv1xRatio=v1y/v1x;
       desiredAngles[i] = theta;
```

```
        desiredAngleRates[i] = 0.0;
        desiredAngles0[i] = desiredAngles[i];
        desiredAngleRates0[i] = desiredAngleRates[i];
      }
      else{
        desiredAngles[i] = desiredAngles0[i];
        desiredAngleRates[i] = desiredAngleRates0[i];
      }
      break;

    case 6:
    case 7:
    case 8:
    case 9:
      if (new_abs(desiredSpeeds[i]) > 0.01){
        desiredAngles[i] = atan2(v1y,v1x) - vehicle.heading;
        wheelAngleV = motion.Theta - vehicle.heading - whp[i].alpha;
        desiredAngleRates[i] =
          ( (speed*speed*thetaDot + ro2*Omega3)
               +speed*ro*omega*(omega+thetaDot)*sin(wheelAngleV)
               +ro*(omegaDot*speed-omega*speedDot)*cos(wheelAngleV) )
            /( desiredSpeeds[i]* desiredSpeeds[i]) - omega;
```

```
            desiredAngles0[i] = desiredAngles[i];
            desiredAngleRates0[i] = desiredAngleRates[i];
        }
        else{
            desiredAngles[i] = desiredAngles0[i];
            desiredAngleRates[i] = desiredAngleRates0[i];
        }
        break;

    case 10:
        desiredSpeeds[i] = speed *
        (new_sqrt((ai[i]*vehicle.kappa)*(ai[i]*vehicle.kappa)
      +(1-bi[i]*vehicle.kappa)*(1-bi[i]*vehicle.kappa)));
        if (vehicle.kappa != 0.0){
            desiredAngles[i] = atan2(bi[i],(vehicle.kappa-ai[i]));
        }
        else { desiredAngles[i] = 0.0;}
        desiredAngleRates[i] =  0.0;
        break;
    case 100:
        break;
}/* end switch */
```

```
  }/* end for */
}



/* ---------------------------------------------------------- *
 * joystickMotionInterface ejm 19 June 97          *     *
 * ---------------------------------------------------------- */
void joystickMotionInterface()
{
  motion.Speed = joyStick.y;   /* convert x-position into double */
  motion.Theta = joyStick.x;   /* convert y-poistion into double */
  motion.Omega =0.0;           /*motion.Omega = joyStick.w; not implemented yet;*/
}

/* ---------------------------------------------------------- *
 * tangentialMotion                                           *
 * ---------------------------------------------------------- */
void tangentialMotion()
{
  double deltaTheta, deltax, deltay, Si, totalDistance, deltaDistance;
  int ix;
```

```
deltax = 0;
deltay = 0;
for (ix = 0; ix < 4; ix++){
  deltax = deltax + actualSpeeds[ix]*cos(actualAngles[ix]);
  deltay = deltay + actualSpeeds[ix]*sin(actualAngles[ix]);
}

/*returns the linear distance the vehicle has travelled */
deltaS = (DeltaT/4)*new_sqrt((deltax*deltax)+(deltay*deltay));

/* returns the difference between the changes in the distance */
/* of the left and right wheels                              */
deltaTheta = 0.0;
for (ix = 0; ix < 4; ix++){
  Si = actualSpeeds[ix]*DeltaT;
  deltaTheta = deltaTheta + (sin(actualAngles[ix])/ai[ix]
                        - cos(actualAngles[ix])/bi[ix])*Si;
}
deltaTheta = deltaTheta/4;

totalDistance += deltaS;  /* Keeps track of the total distance traved by vehicle */
```

```
/* update the vehicle's configuration based on the distance travelled */
/* during the last motion control cycle                               */

vehicle.heading += deltaTheta;
circularArc(deltaS, deltaTheta);
compose();

deltaDistance = DeltaT*motion.Speed;
/*  vehicle.kappa += (steer()*deltaDistance);*/
vehicle.kappa = 0.0;
/*  motion.Theta += deltaTheta;              */
/*  motion.Theta = vehicle.heading;          */
motion.Theta = 0.0;
/*  motion.Omega = vehicle.kappa*motion.Speed;  */
motion.Omega = 0.0;
thetaDot = deltaTheta/DeltaT;
speedDot = 0.0;
omegaDot = 0.0;
}

/**************************************************************/
```

```
/* FUNCTION:  circularArc()                                          */
/* PARAMETERS: Configuration length  --the arc length                */
/*                     alpha   --the end orientation                 */
/*                     config  --pointer to the resultant configuration  */
/* PURPOSE:   Given the arc length and alpha, to calculate the final */
/*            configuration                                          */
/* RETURNS:   Configuration: pointer to the final configuration      */
/* COMMENTS:  The main purpose of this function is to be used in conjunction */
/*            with compose() to form a new next().  In this case, length would*/
/*            actually be delta-s and alpha would be delta-theta.    */
/*            Circular_arc() would determine the configuration after the incre-*/
/*            mental move in the local coordinate system of the original */
/*            configuration.  Then compose() would take the original */
/*            configuration (in global coordinates) and the incremental */
/*                configuration (in local coordinates) to determine the */
/*            incremental configuration in global coordinates.       */
/***********************************************************************/
void
circularArc(double length, double alpha)
{

 double alpha2,alpha4;
```

```
alpha2 =  alpha*alpha;
alpha4 =  alpha2*alpha2;
defineConfig((1- alpha2/6.0 + alpha4/120.0) * length,
             (0.5 - alpha2/24 + alpha4/720.0) * alpha * length,
             alpha, 0.0);
}
/*************************************************************/
/*  FUNCTION:   defineConfig()                                  */
/*  PARAMETERS: double x,y,theta,kappa     --The values that define a   */
/*                         configuration                        */
/*  PURPOSE:    To allocate nad assign a configuration          */
/*  RETURNS:    Configuration:    a configuration               */
/*  COMMENTS:   Was called def_configuration() in MML10         */
/*************************************************************/
void
defineConfig(double x,double y,double theta,double kappa)
{
  incrementalMotion.coord.x = x;
  incrementalMotion.coord.y = y;
  incrementalMotion.heading = theta;
  incrementalMotion.kappa = kappa;
```

```
}


/**********************************************************************/
/* FUNCTION:   compose()                                              */
/* PARAMETERS: Configuration *first  -- pointer to the first configuration      */
/*                    *second -- pointer to the  second configuration  */
/* PURPOSE:    To calculate the composition of the first and second            */
/*          configurations                                            */
/* RETURNS:    Configuration: configuration which is the              */
/*          composition of the first and second configurations        */
/* COMMENTS:   A typical example of the usage of this function is to determine */
/*          the goal position of a configuration in global coordinates.  In */
/*          such an example, the first argument would be the original      */
/*          configuration and the second argument would be the goal        */
/*          configuration in the original configuration's local coordinate */
/*          system.  The resultant third argument would then be the goal   */
/*          configuration in global coordinates.Was called comp() in MML10      */
/* LAST UPDATE: 10/25/94 Chien-Liang Chuang                                     */
/**********************************************************************/
void
compose()
```

```
{

double x,y, theta;
double xx,yy,tt;

holdVehicle.coord.x = vehicle.coord.x;
holdVehicle.coord.y = vehicle.coord.y;
holdVehicle.heading = vehicle.heading;
holdVehicle.kappa   = vehicle.kappa;

x = incrementalMotion.coord.x;
y = incrementalMotion.coord.y;
theta = holdVehicle.heading;


xx = cos(theta) * x - sin(theta) * y +  holdVehicle.coord.x;
yy = sin(theta) * x + cos(theta) * y +  holdVehicle.coord.y;

tt = holdVehicle.heading + incrementalMotion.heading;

vehicle.coord.x = xx;
vehicle.coord.y = yy;
```

```
 vehicle.heading = tt;
 vehicle.kappa = holdVehicle.kappa;
}

/****************************************************************/
/*  FUNCTION : steer(robot,line) PURPOSE : evaluate steering          */
/*  function                                                           */
/****************************************************************/
double steer()

{
 double lambda, angle, dist;

 if (currentPath.config.kappa == 0.0)
    lambda = - currentPath.a * vehicle.kappa
             - currentPath.b * norm(vehicle.heading - currentPath.config.heading)
             - currentPath.c *(-(vehicle.coord.x - currentPath.config.coord.x)
                      * sin(currentPath.config.heading)
                   +(vehicle.coord.y - currentPath.config.coord.y)
                   * cos(currentPath.config.heading));
 else
  {
```

```
    angle = Psi(vehicle.coord, currentPath.center);
    dist = distance(currentPath.center, vehicle.coord);
    if (currentPath.config.kappa > 0.0)
        {
        lambda = - currentPath.a * (vehicle.kappa-currentPath.config.kappa)
            - currentPath.b * norm(vehicle.heading-(angle-HPI))
                - currentPath.c * (currentPath.radius - dist);
        }
    else
        lambda = - currentPath.a * (vehicle.kappa-currentPath.config.kappa)
            - currentPath.b * norm(vehicle.heading-(angle+HPI))
                - currentPath.c * (currentPath.radius + dist);
  }
    return lambda;
}

/**************************************************************/
void constants()
{
  double k;

  k = 1.0/sigma;
```

```
  currentPath.a = 3.0*k;
  currentPath.b = 3.0*k*k;
  currentPath.c = k*k*k;
}
/****************************************************************/
/* Function: Psi_function()                                     */
/* Purpose:  Computes the Psi function of two given points      */
/* Parameters: point p1,p2                                      */
/* Returns: double                                              */
/* Comments:                                                    */
/**********************************************************/
double
Psi(point p1,point p2)

{
 if ( p2.y - p1.y == 0.0 && p2.x - p1.x == 0.0)
   return 0.0;
 else
   return atan2(p2.y - p1.y, p2.x - p1.x);
}


```

```
/****************************************************************/
/* Function: distance()                                         */
/* Purpose:  Computes the  distance between two given points    */
/* Parameters: point p1,p2                                      */
/* Returns: double                                              */
/* Comments:                                                    */
/****************************************************************/
double
distance(point p1,point p2)

{
  double X, Y;

  X = p1.x - p2.x;
  Y = p1.y - p2.y;
  return new_sqrt( X*X  +  Y*Y );
}

void initTangent()
{
   currentPath.config.coord.x = 0.0;
   currentPath.config.coord.y = 0.0;
```

```
currentPath.config.heading = 0.0;
currentPath.config.kappa = 0.0;
currentPath.radius = 0.0;
currentPath.center.x = 0.0;
currentPath.center.y = 0.0;
sigma = 20.0;
constants();

motion.Speed = 10.0;
motion.Theta = 0.0;
motion.Omega = 0.0;

vehicle.coord.y = 0.0;
vehicle.coord.x = 0.0;


}

```

APPENDIX E: SOURCE CODE (MOTOR.C)

The following code was modified by: Professor Kanayama, Thorsten Leonardy, Edward Mays, and Ferdinand A. Reid.

```
/* ==============================================================
// Edward Mays
// Shpeherd project
// 20 February 1997
// MotionControl
// ==============================================================*/

#include "motor.h"
#include "shepherd.h"
#include "math.h"

void readEncoders() {
  readDriveEncoders(driveReadings);
  readSteerEncoders(steerReadings);
}

/*                                                */
/* Verifies validity of incoming speeds/angles and converts   */
/* digitial input for the DA board                  */
/*                                                */
void driveMotors(){

   int ix,Speed_Digit,Steer_Digit, counter;
```

```
double speed1, steer1, temp;

unsigned short bitMask=0x8000;      /* access bit 15 for align  wheel 1 */
unsigned short *servoStatus=(unsigned short *)(VME9421+0x00ca); /* digital input */

bitMask = bitMask >> 3;

/*  updateWheelDrive();  wheel values for driving              */
/*  updateWheelSteer();                              */
/*  comupte the current actual wheel direction in WheelDirAct[] */

if (mode != 100){
 for(ix =0; ix <ARRAY_SIZE; ix++){
   /* *******************steering/driving interaction*************      */
   /* here +/- 1/50 of the steering value is added to the driving       */
   /* for each specified wheel. Note the negative sign on elements [1]  */
   /* and [3]provide the same direction driving as elements [0] and [2] */

   Omega_Speed = desiredSpeeds[ix] +
    SteerDriveInteract*desiredAngleRates[ix]*WheelRadius; /* cm/sec */

   /* conversion to digits   */
   Speed_Digit = velocityReferenceTable(Omega_Speed,ix) +
       DriveFeedBackGain*(Omega_Speed - actualSpeeds[ix]);
   Steer_Digit = rateReferenceTable(desiredAngleRates[ix])
       + steerFeedbackGain*(desiredAngleRates[ix]-actualAngleRates[ix])
```

```
        + angleFeedbackGain*norm(desiredAngles[ix]-actualAngles[ix]);

    if (Speed_Digit>DigitsHigh)        /* Limitation  */
      Speed_Digit= DigitsHigh;
    if (Steer_Digit>DigitsHigh)
      Steer_Digit= DigitsHigh;
    if (Speed_Digit<DigitsLow)
      Speed_Digit= DigitsLow;
    if (Steer_Digit<DigitsLow)
      Steer_Digit= DigitsLow;

    switch(mode){
        case 2:
        case 3:
        case 4:
        case 5:
        case 6:
        case 7:
        case 8:
        case 9:
        case 10:
          Speed_Digits[ix]= (short)Speed_Digit;  /* casting to short */
          Steer_Digits[ix]= (short)Steer_Digit;
          break;

        case 1:
```

```
            speed1 = Speed_Digits[ix];
            steer1 = Steer_Digits[ix];
            if ( speed1 > 0) speed1--;
            if ( speed1 < 0) speed1++;
            if ( steer1 > 0) steer1--;
            if ( steer1 < 0) steer1++;
            Speed_Digits[ix] = speed1;
            Steer_Digits[ix] = steer1;
            break;
    } /* end switch */
   } /* end for */
  } /* end if */
  else {
    for (ix=0; ix<3; ix++){
            Steer_Digits[ix] = 0;
        }
        for (ix=0; ix<4; ix++){
            Speed_Digits[ix] = 0;
        }

        switch(modeTstate){
          case 0:
            Steer_Digits[3] = 50*Flag;
            modeTstate = 1;
            break;

```

```
case 1:
  modeTstate = 2;
  break;
case 2:
  modeTstate = 3;
  break;
case 3:
  modeTstate = 4;
  break;
case 4:
  modeTstate = 5;
  break;
case 5:
  modeTstate = 6;
  break;
case 6:
  modeTstate = 7;
  break;
case 7:
```

```
    modeTstate = 8;
    break;

  case 8:
    modeTstate = 9;
    break;

  case 9:
    modeTstate = 10;
    break;

  case 10:
    modeTstate = 11;
    break;

  case 11:
    modeTstate = 12;
    break;

  case 12:
    modeTstate = 13;
    break;

  case 13:
    modeTstate = 14;
    break;
```

```
case 14:
  modeTstate = 15;
  break;

case 15:
  modeTstate = 16;
  break;

case 16:
  modeTstate = 17;
  break;

case 17:
  modeTstate = 18;
  break;

case 18:
  modeTstate = 19;
  break;

case 19:
  if (bitMask&*servoStatus)/* read servo status, */
      {                /*wait until wheel aligned */
    Flag = -Flag;
    modeTstate = 20;
```

```
            }
        break;

      case 20:
        Steer_Digits[3] = 0;
        modeTstate = 21;
        break;

      case 21:
        modeTstate = 22;
        break;

      case 22:
        modeTstate = 23;
        break;

      case 23:
        modeTstate = 24;
        break;

      case 24:
        modeTstate = 25;
        break;

      case 25:
        modeTstate = 26;
```

```
            break;

          case 26:
            modeTstate = 27;
            break;

          case 27:
            modeTstate = 0;
            break;

          default :  break;
        } /* end switch */
    } /* end else */

  driveSteer(Steer_Digits);
  driveSpeed(Speed_Digits);
}/* end driveMotors */

/* Wheel Driving function */
void driveSpeed(short Speed_Digits[]) {

unsigned int   *servoControl=(unsigned int *)VME2170;  /* Data Out       */
short *servoOut1=(unsigned short*)(VME9210+0x0082);   /* Analog out     */
short *servoOut3=(unsigned short*)(VME9210+0x0086);   /* Analog out test*/
short *servoOut2=(unsigned short*)(VME9210+0x0084);   /* Analog out test*/
short *servoOut4=(unsigned short*)(VME9210+0x0088);   /* Analog out test*/
```

```
unsigned int wheelSelect=0x00924924; /* select all wheels for driving or steering */
*servoControl=wheelSelect;
*servoOut1= (-Speed_Digits[0])<<4;
*servoOut3= (-Speed_Digits[2])<<4;
*servoOut2=  Speed_Digits[1] <<4;
*servoOut4=  Speed_Digits[3] <<4;
 return;
} /* driveSpeed */
/* Wheel Steering function */
void driveSteer(short Steer_Digits[]){
 unsigned int *servoControl=(unsigned int *)VME2170; /* Data Out      */
 short *servoOut1=(unsigned short*)(VME9210+0x008A); /* Analog out wheel1*/
 short *servoOut3=(unsigned short*)(VME9210+0x008E); /* Analog out wheel3*/
 short *servoOut2=(unsigned short*)(VME9210+0x008C); /* Analog out wheel2*/
 short *servoOut4=(unsigned short*)(VME9210+0x0090); /* Analog out wheel4*/
```

```
/* select all wheels for driving or turning */
unsigned int wheelSelect=0x00924924;

    *servoOut1= Steer_Digits[0]<<4;  /* a neg volt turns wheels clockwise */
    *servoOut3= Steer_Digits[2]<<4;  /* a pos volt turns counter clockwise*/
    *servoOut2= Steer_Digits[1]<<4;
    *servoOut4= Steer_Digits[3]<<4;
    *servoControl=wheelSelect;          /* turn on selected servo motor */

return;

} /* end of driveSteer */



/* Wheel stop  function */
void allStop(){

unsigned int   *servoControl=(unsigned int *)VME2170; /* Data Out */
/* short *servoOut1=(unsigned short*)(VME9210+0x0084);*/

/* deselect all wheels for driving and/or turning */

*servoControl=0x00000000;        /* turn off selected servo motor */
/* *servoOut1= 0.0;*/ /* temp, does not belong in this function */
initBoards();
```

```
 return;

} /* end of allStop */


void wheelDrive()
{ int ix,a;
double alpha[ARRAY_SIZE]={30, 0, 0, 0}, beta[ARRAY_SIZE]={0.0, 0.0, 0.0, 0.0};

driveMotors(alpha,beta);

return;
}/*end wheelDrive */



void readDriveEncoders(unsigned long int array[])
{
 unsigned char *p=(unsigned char*)VMECTR1, c1, c2, c3;
 int ix;
 long int temp;

 for (ix=0; ix<4; ix++) {   /* read all four motors subsequentially */

    *(p+3)=0x03;          /* load  output latch from counter */
    *(p+3)=0x01;          /* control register, initialize two-bit output latch */
```

```
    /* read three bytes for specific counter ix and save in status   */
    /* first access to Output Latch Register reads least significant */
    /* byte first                                                    */

    c1 = *(p+1) & 0x00ff;
    c2 = *(p+1) & 0x00ff;
    c3 = *(p+1) & 0x00ff;
    array[ix] = ((unsigned int)c1)| ((unsigned int)c2 << 8) |
         ((unsigned int)c3 << 16);

    p=p+4;                  /* increment pointer for next counter */


  }
  return;
} /* end of readDriveEncoders */


int readSteerEncoders(unsigned long int array[])
{
  unsigned char *p=(unsigned char*)(VMECTR1 + 0x0100), c1, c2, c3;
  int ix;


  for (ix=0; ix<4; ix++) {   /* read all four motors subsequentially */
```

```
    *(p+3)=0x03;          /* load  output latch from counter */
    *(p+3)=0x01;          /* control register, initialize two-bit output latch */


/* read three bytes for specific counter ix and save in status */
/* first access to Output Latch Register reads least significant byte first */

    c1 = *(p+1) & 0x00ff;
    c2 = *(p+1) & 0x00ff;
    c3 = *(p+1) & 0x00ff;
    array[ix] = ((unsigned int)c1)| ((unsigned int)c2 << 8) |
          ((unsigned int)c3 << 16);


    p=p+4;                         /* increment pointer for next counter */

  }
  return;
} /* end of readSteerEncoders */

void displayDirections()
{
    /*if (edCounter%10 == 0){      */
    convertBCD(bcdString+9,(unsigned int)(WheelDirDes[2]*RadsToDegrees));
                                /*bcdString+9*/
```

```
	convertBCD(bcdString+9,(unsigned int)edCounter);
	bcdString[3]='0';
	bcdString[4]='3';
	bcdString[6]='4';
	bcdString[7]='0';
	sioOut(0,bcdString);

	/* sioOut(0,clrLine); */ /* clear line	*/

	convertBCD(bcdString+9,(unsigned int)(WheelDirAct[2]*RadsToDegrees));
	bcdString[3]='0';
	bcdString[4]='3';
	bcdString[6]='6';
	bcdString[7]='0';
	sioOut(0,bcdString);

}

/* 2 May */
void displaySpeed()
{

	convertBCD(bcdString+9,(unsigned int)1);
	bcdString[3]='0';
	bcdString[4]='3';
	bcdString[6]='4';
```

```
    bcdString[7]='0';
    sioOut(0,bcdString);

    /*  sioOut(0,clrLine); */ /* clear line      */

    convertBCD(bcdString+9,(unsigned int)steerReadings[1]);
    bcdString[3]='0';
    bcdString[4]='3';
    bcdString[6]='6';
    bcdString[7]='0';
    sioOut(0,bcdString);
}

void testDrive1()
{
  desiredAngleRates[0] =  1;
  desiredAngleRates[1] =  1;
  desiredAngleRates[2] =  1;
  desiredAngleRates[3] =  1;
  desiredSpeeds[0] =  0; /* wheels 2&4 must have minus sign */
  desiredSpeeds[1] =  0; /* wheels 2&4 must have minus sign */
  desiredSpeeds[2] =  0; /* wheels 2&4 must have minus sign */
  desiredSpeeds[3] =  0; /* wheels 2&4 must have minus sign */

}

```

```
/* 2 May */
void testDrive()
{
 double  MM, RES, N= 1044548, C=0.001;
sioOut(0,"Entering testDrive ...\n\r");
if(20<(N - steerReadings[1])*C){
   MM=20.0;
 }
 else{
   MM=(N - steerReadings[1])*C;
 }
  if(MM>0){
   RES=MM;
 }
 else{
   RES=0.0;
 }
```

```
/* RES=max(min(20,(N - steerReadings[0])*C),0); */
sioOut(0,"Leaving testDrive ...\n\r");
desiredSpeeds[1] = -(RES); /* wheels 2&4 must have minus sign */
return;
}
void computeActualRates()
{
int i;
double count,speed;
for(i=0; i<=3; i++)
 {
 if(PreviousCountSpeed[i] == 99999999) /* for derivative for speed */
  actualSpeeds[i]= 0.0;
else
```

```
 actualSpeeds[i]=
   (convertDifference((driveReadings[i] - PreviousCountSpeed[i]))
   *DigitToCmDrive[i])/DeltaT;
PreviousCountSpeed[i] = driveReadings[i];

if(PreviousCountSteer[i] == 99999999)  /* for derivative for steering  */
 actualAngleRates[i]= 0.0;
else
 actualAngleRates[i]=
    (convertDifference((steerReadings[i] - PreviousCountSteer[i]))
   *digitToRadSteer)/DeltaT;
PreviousCountSteer[i] =  steerReadings[i];
 }
}



void accumulateDriveSpeed()
{
int i;

for(i=0;i<=3;i++){
 Display_Speeds[i] += actualSpeeds[i];
 }
return;
}
```

```
void accumulateDriveSteer()
{
int i;
for(i=0;i<=3;i++){
  Display_Steers[i] += 10*actualAngleRates[i];
  actualAngles[i]  += actualAngleRates[i]*DeltaT;
}
return;
}
/* added 15 may  */
void displayDriveAngle()
{
double angle, angle1, angle2, angle3;
angle  = actualAngleRates[0] * 1000.0;
angle1 = actualAngleRates[1] * 1000.0;
angle2 = actualAngleRates[2] * 1000.0;
angle3 = actualAngleRates[3] * 1000.0;
 if (edCounter%100 == 0){
  convertInt(bcdString+9,(int)desiredAngleRates[0]);
    bcdString[3]='0';
    bcdString[4]='3';
```

```
  bcdString[6]='4';
  bcdString[7]='0';
  sioOut(0,bcdString);
convertInt(bcdString+9,(int) angle);
  bcdString[3]='0';
  bcdString[4]='3';
  bcdString[6]='6';
  bcdString[7]='0';
  sioOut(0,bcdString);
convertInt(bcdString+9,(int) angle1);
  bcdString[3]='0';
  bcdString[4]='4';
  bcdString[6]='6';
  bcdString[7]='0';
  sioOut(0,bcdString);
convertInt(bcdString+9,(int) angle2);
  bcdString[3]='0';
  bcdString[4]='5';
  bcdString[6]='6';
  bcdString[7]='0';
  sioOut(0,bcdString);
convertInt(bcdString+9,(int) angle3);
  bcdString[3]='0';
  bcdString[4]='6';
  bcdString[6]='6';
  bcdString[7]='0';
```

```
    sioOut(0,bcdString);}

return;
}

double velocityReferenceTable(double desiredVelocity,int i)
{
  double inVelocity,
         outVelocity;

  inVelocity=new_abs(desiredVelocity);

   if (inVelocity>=0.0 && inVelocity<=5.0)
    outVelocity = inVelocity*K1[i];

   if (inVelocity>5.0 && inVelocity< 8.0)
    outVelocity = inVelocity*K2[i];

   if (inVelocity>=8.0 && inVelocity<20.0)
    outVelocity = inVelocity*K3[i];

   if (inVelocity>=20.0 && inVelocity<= 70.0)
    outVelocity = inVelocity*K4[i];
```

```
if (inVelocity>70.0 && inVelocity<K5)
    outVelocity = inVelocity*K6[i];
  if (inVelocity> K5)
   outVelocity=1023;
  if (desiredVelocity< 0.0)
    outVelocity = - outVelocity;
 return outVelocity;
} /* end velocityLookupTable */
double rateReferenceTable(double desiredRate)
{
 double inRate,
        outDigit;
 /*outDigit = new_abs(desiredRate); *//* test only */
  inRate=new_abs(desiredRate);
  if (inRate<= 5.234)
    outDigit = inRate*195.4155 ;
  else
     outDigit=1023;
```

```
 if (desiredRate< 0.0)
   outDigit = - outDigit;

 return outDigit;
}



/*************************************************************
 Function convertDifference() returns the difference between the new shaft
 encoder position and the old shaft encoder position. The shaft encoder values
 contain only 24 bits (0x000000-0xffffff). The routine adjusts for the trans-
 ition from 0xffffff to 0x000000 and vice versa.
*************************************************************/

int convertDifference(int value)
{
 if(value < -0x800000)
   value &= 0x00ffffff;
 else  if(value >= 0x800000)
   value |= 0xff000000;

 return value;
}
```

```
/* ------------------------------------------------------------------------ *
 *                                                                          *
 * File:         S E R V O . C                                              *
 *                                                                          *
 * Environment:  GCC Compiler v2.7.2                                        *
 * Last update:  30 January 1997                                            *
 * Name:         Thorsten Leonardy                                          *
 * Purpose:      Provides the kernel for SHEPHERD.                          *
 *                                                                          *
 * Compiled:     >gcc -c -m68040 -o servo.o servo.c                         *
 *                                                                          *
 * ------------------------------------------------------------------------ */



/* ------------------------------------------------------------------------ *
 * readWheelStatus()                                                        *
 *                                                                          *
 * Environment:  GCC Compiler v2.7.2                                        *
 * Last update:  20 February 1997                                           *
 * Name:         Thorsten Leonardy                                          *
 * Purpose:      This function reads the wheels counter status.             *
 *               This routine makes use of the fact that arrays are stored  *
 *               in memory consecutively.                                   *
 *                                                                          *
```

```
 * array        points to the beginning of the array 'wheelEncoder'.        *
 * ------------------------------------------------------------------------ */
void readWheelStatus(unsigned char *array)
{
  unsigned char *p=(unsigned char*)VMECTR1;
  int ix;

  for (ix=0; ix<8; ix++) {   /* read all eight motors subsequentially */

    *(p+3)=0x03;          /* load  output latch from counter */
    *(p+3)=0x01;          /* control register, initialize two-bit output latch */

    /* read three bytes for specific counter ix and save in status */
    /* first access to Output Latch Register reads least significant byte first */
    *(array+3)=*(p+1);
    *(array+2)=*(p+1);
    *(array+1)=*(p+1);
    *(array+0)=0;

    array+=4;                          /* point to next entry in wheelEncoder*/
    p=p+4;                            /* increment pointer for next counter */

    if (ix==3) p=(unsigned char*)VMECTR2;  /* access the second VME Counter */

  }
 return;
```

```
} /* end of readWheel Status */


/* ---------------------------------------------------------------------- *
 * clearShaftEncoder(unsigned short motors)                                *
 *                                                                         *
 * Environment:  GCC Compiler v2.7.2                                       *
 * Last update:  04 March 1997                                             *
 * Name:         Thorsten Leonardy                                         *
 * Purpose:      This function clears the selected shaft encoder.          *
 *                                                                         *
 * motors        bit mask to select motors, eg. 0x042 selects motor 2 and 7 *
 *               to be cleared.                                            *
 * ---------------------------------------------------------------------- */
void clearShaftEncoder(unsigned short motors)
{
  unsigned char *p=(unsigned char*)VMECTR1;
  int ix;

  for (ix=0; ix<8; ix++,motors/=2) {
    if (motors & 0x01) *(p+3)=0x04;      /* clear respective counter */
    p=p+4;                               /* access next pointer      */
    if (ix==3) p=(unsigned char*)VMECTR2;  /* access the second VME Counter */
  }
  return;
} /* end of clearShaftEncoder */
```

```


/* ------------------------------------------------------------------------ *
 * align()                                                                  *
 * Environment:  GCC Compiler                                               *
 * Last update:  07 August 1997                  m                          *
 * Name:         Thorsten Leonardy ,Yutaka Kanayama, Ed Mays *
 * Purpose:      This function will align SHEPHERD's wheels such that all   *
 *               will point in the forward direction. It utilizes the hall  *
 *               sensors for each of the four wheels. Crucial parameters    *
 *               are as follows:                                            *
 *                                                                          *
 * ------------------------------------------------------------------------ */
void align(void)
{
  unsigned int   *servoControl=(unsigned int *)VME2170;          /* Data Out */
  unsigned short *servoOut=(unsigned short*)(VME9210+0x008A);    /* Analog out */
  unsigned short *servoStatus=(unsigned short *)(VME9421+0x00ca); /* digital input */
  unsigned short bitMask=0x8000, bitMask1;     /* access bit 15 for align  wheel 1 */
  unsigned int wheelSelect=0x00004000; /* select servo for turning wheel 1 */
  int ix, notYet;                       /* just a counter */

  do {
   notYet = 0;
   bitMask1 = bitMask;
   for (ix=0; ix < 4; ix++)
```

```
    {
       if (bitMask1 & *servoStatus)
         {
           Steer_Digits[ix] = 0;
         }
       else
         {
           Steer_Digits[ix] = 40;
           notYet++;
         }
       bitMask1 = bitMask1 >> 1;     /* select next status align bit    */
    }
  driveSteer(Steer_Digits);
  } while(notYet);
 *servoControl=0x00000000;         /* disable all wheels              */
 return;
} /* end of align */

/* ---------------------------------------------------------------------- *
 * alignAfterRotate()                                                     *
 * Environment:  GCC Compiler                                             *
 * Last update:  07 August 1997                 m                         *
 * Name:         Thorsten Leonardy,Yutaka Kanayama, and Ed Mays *
 * Purpose:      This function will align SHEPHERD's wheels such that all  *
 *            will point in the forward direction. It utilizes the hall     *
 *            sensors for each of the four wheels. Crucial parameters       *
```

```
*            are as follows:                                    *
*                                                               *
* ------------------------------------------------------------------------- */
void alignAfterRotate(void)
{
  unsigned int   *servoControl=(unsigned int *)VME2170;        /* Data Out */
  unsigned short *servoOut=(unsigned short*)(VME9210+0x008A);    /* Analog out */
  unsigned short *servoStatus=(unsigned short *)(VME9421+0x00ca); /* digital input */
  unsigned short bitMask=0x8000, bitMask1;       /* access bit 15 for align  wheel 1 */
  unsigned int wheelSelect=0x00004000; /* select servo for turning wheel 1 */
  int ix, notYet;                         /* just a counter */

  do {
   notYet = 0;
   bitMask1 = bitMask;
   for (ix=0; ix < 4; ix++)
     {
         if (bitMask1 & *servoStatus)
           {
            Steer_Digits[ix] = 0;
           }
         else
           {
            if( ix==1 || ix==2 )
              Steer_Digits[ix] = 40;  /* for wheel 1 and 2, rotate CCW */
            else
```

```
            Steer_Digits[ix] = -40;  /* for wheel 0 and 3, rotate CW  */
           notYet++;
         }
       bitMask1 = bitMask1 >> 1;     /* select next status align bit    */
   }
  driveSteer(Steer_Digits);
  } while(notYet);
 *servoControl=0x00000000;          /* disable all wheels            */
 return;
} /* end of align */


/* ----------------------------------------------------------------------- *
 * alignWheels()                                          *
 * Environment:  GCC Compiler                                      *
 * Last update:  07 January 1997                                 *
 * Name:        Thorsten Leonardy                                *
 * Purpose:      This function will align SHEPHERD's wheels such that all    *
 *              will point in the forward direction. It utilizes the hall  *
 *              sensors for each of the four wheels. Crucial parameters     *
 *              are as follows:                             *
 *                                                    *
 * servoControl  Base address for the channels controling the servo motors   *
 *              switch servos on an off by accessing this address.        *
 *              Each servo is controlled by three bits:            *
 *              bits 0..2  -> driving wheel 1                   *
```

```
*              3..5  -> driving wheel 2                 *
*              6..8  -> driving wheel 3                 *
*              9..11 -> driving wheel 4                 *
*             12..14 -> turning wheel 1                 *
*             15..17 -> turning wheel 2                 *
*             18..20 -> turning wheel 3                 *
*             21..23 -> turning wheel 4                 *
*             24..31 -> not used                        *
*                                                       *
* servoOut    Base address for the analog output card controlling the  *
*           speed of the servos. Only the highest 12 bits are used.   *
*           0x0010 -> selects lowest positive velocity          *
*           0x7ff0 -> selects highest positive velocity         *
*           0xfff0 -> selects lowest negative velocity (i.e -1 m/s)   *
*           0x8000 -> selects highest negative velocity (i.e. -1000m/s) *
*                                                       *
*           It has been found that the MSB does not work properly.   *
*           Therefore, the velocities should lie within 11 bit range,  *
*           -1024 <= velocity <=  +1023                 *
*                                                       *
* servoStatus  Base address for reading the servo status          *
*           The alignment bits are: Port B, Bit 15 for wheel 1     *
*                                   Bit 14 for wheel 2   *
*                                   Bit 13 for wheel 3   *
*                                   Bit 12 for wheel 4   *
*                                                       *
```

```
* ------------------------------------------------------------------------- */


void alignWheels(void)
{
  unsigned int   *servoControl=(unsigned int *)VME2170;          /* Data Out */
  unsigned short *servoOut=(unsigned short*)(VME9210+0x008A);    /* Analog out */
  unsigned short *servoStatus=(unsigned short *)(VME9421+0x00ca); /* digital input */

  unsigned short bitMask=0x8000;      /* access bit 15 for align  wheel 1 */
  unsigned int wheelSelect=0x00004000; /* select servo for turning wheel 1 */
  int wheel;                          /* just a counter */

  /* -------------------------------------------------------------- *
   * align wheels subsequentially, start with wheel 1 (front right) *
   * -------------------------------------------------------------- */
  for (wheel=1; wheel<5; wheel++) {

    *servoOut++=0x0200;              /* set output value for servo first          */
                                     /* 0x0010 corresponds to smallest velocity   */
    *servoControl=wheelSelect;       /* turn on selected servo motor              */
    while(!(bitMask&*servoStatus));  /* read servo status, wait until wheel aligned */
    wheelSelect= wheelSelect<<3;     /* select next servo (motor)                 */
    bitMask = bitMask >> 1;          /* select next status align bit              */
  }

```

```
*servoControl=0x00000000;        /* disable all wheels                    */

/* clearShaftEncoder(0x0ff); */       /* clear all shaft encoders                  */

/* sioOut(0,"aligned ..."); */       /* Output Message                          */
return;
} /* end of alignWheels */




/***************************************************************************
End of servo.c
***************************************************************************/

```

APPENDIX F: SOURCE CODE (TIMER.C)

```
/* ---------------------------------------------------------------------- *
 *                                                                        *
 * File:        T I M E R . C                                             *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Last update:  29 January 1997                                          *
 * Name:        Thorsten Leonardy                                         *
 * Purpose:     Provides routines related to the AM9513 Timer Circuit, such *
 *              as interrupt initialization,                              *
 * Compiled:    >gcc -c -m68040 -o timer.o timer.c                        *
 *                                                                        *
 * ---------------------------------------------------------------------- */


#include "shepherd.h"
#include "timer.h"


void timerStart(void)
{

  long *vadr;
  unsigned char *p;
  short *ctrlPort = (short*) TIMER_CTRL;
  short *dataPort = (short*) TIMER_DATA;

  /* initialize the interrupt counter */
  intCounter=0;
```

```

/* load address for interrupt service routine */
vadr=(long*)VBA_TIMER;
*vadr=(long)TimerHandler;

/* Issue commands to set control and data register              */
/* refer to Fig 1-20, 1-8, 1-12                           */

*ctrlPort=0xffff;        /* Master reset, clear data registers */
*ctrlPort=0xff5f;        /* load all counters              */
*ctrlPort=0xffef;        /* Set MM13 (Enter 16-bit bus mode)  */

*ctrlPort=0xff17;        /* Select master mode register       */
*dataPort=0xa1e0;        /* set master mode register ...      */
/*      +------------>  f == 1 sec interrupt interval    */
/*                      e == 0.1 sec              */
/*                      d == 0.01 sec             */
/*                      c == 0.001 sec            */

*ctrlPort=0xff05;        /* Select CMR timer 5 ...         */
                    /* utilize Data Pointer Sequencing   */

*dataPort=0x0e32;        /* and write to counter mode register */
/*      +------------->              f = 10000     */
/* multiply value according to dataPort below   e = 1000      */
/* by the factor set here to obtain timing...  d =  100     */
/*                                   c =  10     */
/*                                   b =  1     */

```

```
   /* to obtain the correct timing, multiply value determined in data-*/
   /* port below by the factor given above. E.g. dataPort is set to  */
   /* 58 (corresponding to 10usec) and factor 1000 is chosen above,  */
   /* then the interrupt would occur every 10msec!                   */

   *dataPort=58;          /* load register, 58 -> 10usec     */
                          /*                930 -> 1msec     */

   *ctrlPort=0xff70;      /* load and arm timer 5            */

   p=(unsigned char*)ISM_TIMER; /* ISM Configuration for Timer     */
   *p=0xcb;               /* assert LIRQ-3 to VIC            */

   p=(unsigned char*)VIC_LIRQ3; /* VIC LICR for LIRQ-3 from ISM    */
   *p=0x03;               /* assert IRQ-3 from VIC to 68040  */

   return;
 }  /* end of timerStart */


/*****************************************************************
  Assembler routines
 *****************************************************************/

/* TimerHandler, its address is set from within timerStart */

  asm("
     .even
     .text
```

```
        .globl _TimerHandler
_TimerHandler:
        link      a6,#-184            /* alocate 184 Bytes on stack to save registers    */
        fsave     a6@(-184)
        fmovemx   fp0-fp7,sp@-        /* move floating point registers 80 bit each        */
        fmovel    fpcr,sp@-           /* move floating point Control Regioster           */
        fmovel    fpsr,sp@-           /* move floating point status register             */
        fmovel    fpiar,sp@-          /* move floating point Instruction address register */
        moveml    d0-d7/a0-a5,sp@-    /* save data and address registers (14*4 Byte)      */
        addq.l    #0x01,_intCounter   /* increment interrupt counter                     */
        move.w    #0xffe5,0xfff41002  /* clear toggle out for timer 5                    */
        move.l    #0xffff0081,a1      /* load VME9421 Status register                    */
        eor.b     #0x02,(a1)          /* toggle green indicator light to indicate timer  */
                                      /* for interrupt handling is working properly ...  */
        and.b     #0xfe,(a1)          /* turn red light on to indicate that motion control*/
                                      /* will start (this will assert the SYSFAIL line on */
                                      /* the VME-Bus, but we don't care at this point).  */
        jsr _driver                   /* execute motion control part                     */
        move.l    #0xffff0081,a1      /* load VME9421 (digital out board) Status register */
        or.b      #0x01,(a1)          /* turn off red indicator light to indicate that   */
```

```
                        /* motion control is done.                   */

    moveml  sp@+,d0-d7/a0-a5
    fmovel  sp@+,fpiar
    fmovel  sp@+,fpsr
    fmovel  sp@+,fpcr
    fmovemx sp@+,fp0-fp7
    frestore a6@(-184)
    unlk    a6

    rte
  ");


/************************************************************************
 End of timer.c
 ************************************************************************/

```

APPENDIX G: SOURCE CODE (MATH.C)

The following code was modified by: Professor Kanayama, Thorsten Leonardy, Edward Mays, and Ferdinand A. Reid.

```
/* ---------------------------------------------------------------------- *
 *                                                                        *
 * File:        M A T H . C                                               *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Last update:  17 March 1997                                            *
 * Name:        Thorsten Leonardy                                         *
 * Purpose:     A Simple Math library.                                    *
 * ---------------------------------------------------------------------- */

#include "shepherd.h"
#include "math.h"

#define pio4    0.785398163
#define pio2    1.570796327
#define pi      3.141592654
#define pi2     6.283185307

```

```
/*********************************************************************
 FUNCTION:  norm()
 PARAMETERS: double angle    ---- the angle to normalize
 PURPOSE:   normalize the input angle between -PI and PI
 RETURNS:   double:  the normalized angle in radians
 COMMENTS:  This is the most common normalizing function used in the system
            This performs that same as norm() and normalize)() in MML10.
*********************************************************************/
double norm(double angle)
{
 while ((angle > pi) || (angle <= -pi))
   {
    if (angle > pi)
      angle -= pi2;
    else
      angle += pi2;
   }
 return angle;
}
```

```
/* ------------------------------------------------------------------- *
 *                                                                      *
 * new_abs()                                                            *
 *                                                                      *
 * Environment:  GCC Compiler v2.7.2                                    *
 * Last update:  14 March 1997(mod 2 April 97 by Ed Mays)               *
 * Name:         Thorsten Leonardy                                      *
 * Purpose:      A function returning the absolute value of x.          *
 * ------------------------------------------------------------------- */
double new_abs(double x)
{
  if  (x>=0.0)
    return (x);
  else
    return(-x);
}
```

```
/* ---------------------------------------------------------------------- *
 * atan2()                                                                *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Last update:  17 March 1997                                            *
 * Name:         Thorsten Leonardy                                        *
 * Purpose:      Computes tan(y/x) where x,y are real. If both variables are *
 *               zero, atan2 returns zero. For any other values, atan2 will *
 *               return the positive angle for the (x,y)-pair, e.g.,      *
 *               (x,y)=(0,-1) would return atan2=3/2*pi !                 *
 *               ix determines the accuracy (highest order term in expansion)*
 *               For the worst case, |y/x| close to one, ix should be very *
 *               high. Here is some data:                                 *
 *               |x/y|    ix      accuracy of result [rad]                *
 *                0.9     101      +- 1.88*10E-7                          *
 *                        1001     +- 1.57*10E-49                         *
 *                0.99    101      +- 3.45*10E-3                          *
 *                        1001     +- 4.18*10E-8                          *
 *                        10001    +- 2.18*10E-48                         *
 *                0.999   101      +- 8.76*10E-3                          *
 *                        1001     +- 3.65*10E-4                          *
```

```
 *                    10001      +- 4.50*10E-9                 *
 * ----------------------------------------------------------------------- */
double atan2(double y, double x)
{
   double erg=0.0, z=0.0, z2;
   int ix=101, flag1=0,flag2=0;

   if ((new_abs(y)>new_abs(x))&&(y!=0))
   {
     z=x/y;               /* in case |y/x|>1 compute atan(1/z) */
     flag1=(y>0)-(y<0);     /* a handy sign-function */
   }
   else if (x!=0)
   {
     z=y/x;               /* in case |y/x|<1 compute atan(z)  */
     flag2=(x<0.0);         /* in this case need to add pi to final result */
   }

   /* From here on |z| must always be less than one !!! */
   z2=z*z;

   /* Taylor expansion */
```

```
if (new_abs(z)<1.0) {          /* computation for |y/x|<1 */
  while (ix>1) {
    erg=z2*(1.0/ix-erg); /* try alternatively for accuracy: (z2/ix)*(1.0-ix*erg) */
    ix-=2;
  }
  erg=z-z*erg;
}
else erg=((z>0.0)-(z<0.0))*pio4;  /* for |y/x|=1 result is either +- pi/4 */
if (flag1==1) erg=pio2-erg;       /* point lies in 3rd or 4th octant for flag1=+1 */
else if (flag1==-1) erg=-pio2-erg; /* ... or in 6th or 7th octant for flag1=-1    */
if (flag2) erg=erg+pi;            /* point lies in 4th or 5th octant          */
/* if (erg<0.0) erg=erg+pi2;      deleted 6/27/97                           */
return(erg);
}
/* ----------------------------------------------------------------------*
 * atan()    yk                                        *
 * ----------------------------------------------------------------------*/
double atan(double x)
```

```
{
  double erg=0.0, z=0.0, z2;
  int ix= 101, flag=0;

  if (x == 0.0)
   return (0.0);
  if (new_abs(new_abs(x)-1.0) < 0.00001)
   return (pio4 * x);      /* return +- pi/4          */
  if (new_abs(x) > 1.0)
  {
    z=1.0/x;               /* in case |x|>1 compute atan(1/x) */
    flag=(x>0)-(x<0);      /* a handy sign-function */
  }
  else
    z=x;                   /* in case |x|<1 compute atan(x)  */
  z2=z*z;           /* From here on |z| is less than one !!! */
  /* Taylor expansion */
  while (ix>1)
   {
        erg=z2*(1.0/ix-erg);
        ix-=2;
   }
```

```
 erg=z-z*erg;
 if (flag ==  1) erg = pio2-erg;
 if (flag == -1) erg =-pio2-erg;
 return(erg);
}



/* ---------------------------------------------------------------------- *
 * cos()                                                      *
 *                                                            *
 * Environment:  GCC Compiler v2.7.2                                  *
 * Last update:  17 March 1997                               *
 * Name:         Thorsten Leonardy                           *
 * Purpose:      Computes cos(x) where x can be any real number.         *
 *               ix determines the accuracy (highest order term in expansion)*
 * ---------------------------------------------------------------------- */
double cos(double x)
{
 double erg;
 int quadrant, ix=20;  /* ix must be an even number      */

```

```
/* analyze and reduce x to the appropriate range ... */
quadrant=(x/pio2+(x>=0)-(x<0))/2; /* determine in what sector x is    */
x=x-quadrant*pi;                /* reduce x to region [-pi/2...pi/2]*/
x=x*x;                          /* compute x^2 and store in x       */
erg=1.0;

/* the cosine taylor computation is a one-liner ;-) */
while (ix>0) {
  erg=1.0-erg*x/ix/(ix-1);
  ix-=2;
}

/* shift sign if quadrant is not 1,3,5,... */
if (quadrant%2) erg=-erg;

return(erg);
}


/* ----------------------------------------------------------------------- *
 * sin()                                                                   *
 *                                                                         *
```

```
 * Environment:  GCC Compiler v2.7.2                          *
 * Last update:  14 March 1997                                *
 * Name:        Thorsten Leonardy                             *
 * Purpose:     Computes sin(x) where x can be any real number. *
 * ---------------------------------------------------------------------- */
double sin(double x)
{
  return(cos(x-pio2));  /* since sin(x)=cos(x-pi/2) */
}
/* ---------------------------------------------------------------------- *
 * sqrt()                                                     *
 *    Ed Mays and Ferdinand Reid March 1997                   *
 * Environment:  GCC Compiler v2.7.2                          *
 * ---------------------------------------------------------------------- */
double new_sqrt(double x)
 {
  double x1, x2;
  int count;
```

```
 if (x == 1.0) return(1.0);
 x1 = 1.0;
 for (count=0; count < 10; count++){
   x2 = .5 * (x1 + x/x1);
   x1 = x2;
 }

  return (x2);
}



/* ---------------------------------------------------------------------- *
 * new_sqrt1()                                                              *
 *     Ed Mays and Kanayama                                                 *
 * Environment:  GCC Compiler v2.7.2                                        *
 * ---------------------------------------------------------------------- */
double new_sqrt1(double x)
{
 double x1, x2;
```

```
x1 = 1.0;
x2= -1.0;
while(new_abs(x1-x2) < 1.0e-9)
 {
   x2 = x1;
   x1 = .5 * (x2 + x/x2);
  }

return (x1);
}




/* ed move  to math.c*/
double min (double a, double b)
{
 if (a <= b)
   return a;
 else
   return b;
}
```

```
/* ed move to math.c*/
double max (double a, double b)
{
  if (a>=b)
    return a;
  else
    return b;
}
/*ed*/







/***************************************************************************
  End of math.c

****************************************************************************/
```

APPENDIX H: SOURCE CODE (UTILS.C)

```
/* ------------------------------------------------------------------------ *
 *                                                                          *
 * FILE:      U T I L S . C                                                 *
 *                                                                          *
 * ENVIRONMENT:  GCC COMPILER V2.7.2                                        *
 * LAST UPDATE:  03 FEBRUARY 1997                                           *
 * NAME:      THORSTEN LEONARDY                                             *
 * PURPOSE:     PROVIDES THE UTILITY FUNCTIONS FOR
PROGRAM SHEPHERD.                                                           *
 *                                                                          *
 * COMPILED:    >GCC -C -M68040 -O UTILS.O UTILS.C                          *
 *                                                                          *
 * ------------------------------------------------------------------------ */

#INCLUDE "SHEPHERD.H"
#INCLUDE "UTILS.H"
#INCLUDE "MATH.H"


UNSIGNED INT PIFLAG=0;
UNSIGNED INT MAGIC=0X1237;
```

```
EXTERN CHAR JOYSTICK[];    /* DEFINED IN SHEPHERD.C */
EXTERN CHAR BCDSTRING[]; /* DEFINED IN SHEPHERD.C */


/* ---------------------------------------------------------------------- *
 * READCLOCK()                                                    *
 *                                                      *
 * ENVIRONMENT:    GCC    COMPILER    V2.7.2 *
 * LAST UPDATE:  26 FEBRUARY 1997                                      *
 * NAME:      THORSTEN LEONARDY                                         *
 * PURPOSE:    THIS FUNCTION READS THE VALUES FROM THE CALENDAR CLOCK    *
 *            DEVICE MTK48T08 (SEE OMNIBYTE HANDOUT CHAP. 2.9.4) INTO    *
 *            GLOBAL VARIABLE CLOCK. THE FORMAT IN CLOCK IN DECIMAL IS:  *
 *                                                      *
 *            CLOCK = YYMMDDHHMMSS                                     *
 *                                                      *
 *                        I.E TO RETRIEVE THE DATE PERFORM DATE=CLOCK/1000000;    *
 *                            TO RETRIEVE THE TIME PERFORM TIME=CLOCK%1000000;    *
 *                                                      *
 * CALLED BY:     FUNCTION TIMERHANDLER IN 'TIMER.C' *
```

```
*                                             *
* ------------------------------------------------------------------------ */





/* ------------------------------------------------------------------------ *
* PITEST()                                          *
*                                             *
* ENVIRONMENT: GCC COMPILER V2.7.2 *
* LAST UPDATE: 24 FEBRUARY 1997                               *
* NAME:     THORSTEN LEONARDY                               *
* PURPOSE: THIS FUNCTION TESTS INTERPROCESSOR SIGANNLING VIA PI-46 *
*           INTERRUPT.                              *
* ------------------------------------------------------------------------ */
VOID PITEST(VOID)
{
  LONG *VADR;
  UNSIGNED CHAR *P;

    /* SET ADDRESS FOR PROCESSOR INTERRUPT HANDLER ROUTINE */
```

```
VADR=(UNSIGNED LONG *)VBA_PI;
*VADR=(UNSIGNED LONG)PIHANDLER;

P=(UNSIGNED CHAR *)ISM_PI;
 *P=(UNSIGNED CHAR)0XE0;          /* SET IP-ISM TO 68040 ON LIRQ-6 */

P=(UNSIGNED CHAR *)VIC_LIRQ6;
*P=(UNSIGNED CHAR)0X06;        /* CONFIGURE VIC068 LIRQ-6 */

    P=(UNSIGNED CHAR *)APP_ICR;          /* ABORT/PROC INTERRUPT CTRL */
*P = *P | (UNSIGNED CHAR)0X02;  /* ASSERT IP-46 INTERRUPT */

   WHILE(PIFLAG==0) { INTCOUNTER=0; } /* WAIT FOR PI INTERRUPT */

IF (PIFLAG==MAGIC) {
 /*  TOGGLEVME((UNSIGNED CHAR *)VME9210,0X02); */
  SIOOUT(0,"PASSED");
}
ELSE {
 /*  TOGGLEVME((UNSIGNED CHAR *)VME9421,0X02); */
  SIOOUT(0,"FAILED");
```

```
    }

    RETURN;
} /* END OF PITEST */



/* ----------------------------------------------------------------------- *
 * SETVME()                                   *
 *                                         *
 * ENVIRONMENT: GCC COMPILER V2.7.2 *
 * LAST UPDATE: 24 FEBRUARY 1997                          *
 * NAME:      THORSTEN LEONARDY                           *
 * PURPOSE:   THIS FUNCTION OUTPUTS DATA TO THE STATUS REGISTER OF THE *
 *            SPECIFIED VME BOARD.                    *
 * ----------------------------------------------------------------------- */
VOID SETVME(UNSIGNED CHAR *BOARDADDRESS, UNSIGNED CHAR DATA)
{
    BOARDADDRESS = BOARDADDRESS + 0X81;    /* ACCESS STATUS REGISTER */
   *BOARDADDRESS=DATA;              /* WRITE DATA        */
   RETURN;
}
```

```
/* ----------------------------------------------------------------------- *
 * TOGGLEVME()                                                             *
 *                                                                         *
 * ENVIRONMENT:     GCC COMPILER V2.7.2                                    *
 * LAST UPDATE:  24 FEBRUARY 1997                                          *
 * NAME:       THORSTEN LEONARDY                                           *
 * PURPOSE:       THIS FUNCTION PERFORMS AN XOR OPERATION ON THE STATUS    *
 *                REGISTER OF THE SPECIFIED VME BOARD.                     *
 * ----------------------------------------------------------------------- */
VOID TOGGLEVME(UNSIGNED CHAR *BOARD, UNSIGNED CHAR DATA)
{
  BOARD = BOARD + 0X81;   /* ACCESS STATUS REGISTER */
   *BOARD = *BOARD ^ DATA;  /* TOGGLE BIT WITH BITWISE XOR */
  RETURN;
}
```

```
/* ------------------------------------------------------------------------ *
 * INITBOARDS()                                                             *
 *                                                                          *
 * ENVIRONMENT:  GCC COMPILER V2.7.2                                        *
 * LAST UPDATE:  24 FEBRUARY 1997                                           *
 * NAME:         THORSTEN LEONARDY                                          *
 * PURPOSE:      THIS FUNCTION INITIALIZES ALL VME BOARDS.                  *
 * ------------------------------------------------------------------------ */
VOID INITBOARDS(VOID)
{
  UNSIGNED CHAR *P;
  INT IX;

     P=(UNSIGNED CHAR*)VIC_TTR;           /* VIC TRANSFER TIMEOUT REGISTER    */
  *P = 0XFF;                        /* DISBLE ALL WATCHDOGS             */

     P=(UNSIGNED CHAR*)VIC_ICR;           /* VIC INTERFACE CONFIGURATION REG. */
   *P=0X40;                         /* PREVENT DEADLOCKS, THIS IS A MUST! */

     P=(UNSIGNED CHAR*)VME9421+0X81;  /* ACCESS STATUS REGISTER FOR DI    */
```

```
*P = 0X03;                /* DISABLE SYSFAIL SIGNAL, SET GREEN */

P=(UNSIGNED CHAR*)VME9210+0X81;  /* ACCESS STATUS REGISTER FOR DA     */
*P = 0X03;                /* DISABLE SYSFAIL SIGNAL, SET GREEN */


/* ---------------------------------------------------------- *
 * INITIALIZE ALL EIGHT QUADRATURE COUNTERS (WHEEL ENCODER) *
 * ---------------------------------------------------------- */

P=(UNSIGNED CHAR*)VMECTR1;
 FOR (IX=0; IX<8; IX++) {              /* READ MOTORS SUBSEQUENTIALLY  */
  *(P+3)=0X20;             /* CR: MASTER RESET          */
  *(P+3)=0X48;             /* IC: ENABLE COUNTING        */
  *(P+3)=0XC1;             /* QR: COUNT FULL CYCLE       */
  P=P+0X04;                /* ACCESS NEXT COUNTER        */
   IF (IX==3) P=(UNSIGNED CHAR*)VMECTR2; /* ACCESS THE SECOND VME COUNTER */
}

SIOOUT(0,"BOARDS INITIALIZED ...\N\R");
```

```

  RETURN;
}

/* MODIFIED ED MAYS 18 APR 97 */
UNSIGNED CHAR B2A(INT VALUE)
{
  UNSIGNED CHAR CHAR;

  IF (VALUE < 10){
    CHAR = 48 + VALUE;
  }
  ELSE {
    CHAR = 55 + VALUE;
  }
  RETURN CHAR;
}

VOID B2A2(UNSIGNED CHAR *S, UNSIGNED CHAR CC)
{
  INT LOW, HIGH;

  LOW = CC & 0X0F;
  HIGH = CC/16;
```

```

*S = B2A(LOW);
*(S-1) = B2A(HIGH);
}


/****************************************************************************
ASSEMBLER ROUTINES
****************************************************************************/

ASM("
        .EVEN
        .TEXT
        .GLOBL _PIHANDLER

_PIHANDLER:
            MOVE.L #0XFFF4800C,A1        /* LOAD APP-ICR INTO A1 */
            AND.B   #0XFD,(A1)            /* REMOVE PENDING IP-46 INTERRUPT SIGNAL */
            MOVE.L #0X1237,_PIFLAG       /* SET PIFLAG VARIABLE */
        RTE
```

```
");


/* ------------------------------------------------------------------------ *
 * CONVERTTOASCII()                                                         *
 *                                                                          *
 * ENVIRONMENT: GCC COMPILER V2.7.2                                         *
 * LAST UPDATE: 02 MAY 1997                                                 *
 * NAME:        THORSTEN LEONARDY                                           *
 * PURPOSE:     THIS FUNCTION CONVERTS AN UNSIGNED
        INTEGER TO ITS ASCII    *
 *              EQUIVALENT AND WRITES THIS INTO A STRING.
*
 * NDIGITS             NUMBER OF DIGITS TO CONVERT
*
 * DATA         THE INTEGER TO CONVERT                                      *
 * STR          POINTER TO STRING                                           *
 * ------------------------------------------------------------------------ */
VOID CONVERTTOASCII(UNSIGNED INT NDIGITS, UNSIGNED
INT DATA, CHAR *STR)
{
  UNSIGNED INT I;

  STR=STR+NDIGITS-1;
  FOR (I=0;I<NDIGITS;I++) {
```

```
    *STR-- = '0' + DATA %10 ;
    DATA=DATA/10;
  }
  RETURN;
}


/* ------------------------------------------------------------------------ *
 * READJOYSTICK()                                         *
 *                                         *
 * ENVIRONMENT:    GCC    COMPILER    V2.7.2 *
 * LAST UPDATE:  02 MAY 1997                                *
 * NAME:     THORSTEN LEONARDY                                 *
 * PURPOSE:    THIS FUNCTION READS THE THREE PORTS (A,B AND C) FROM THE   *
 *             INTEL 85C55 PARALLEL PORT 1 AND CONVERTS THEM INTO AN ASCII *
 *          STRING.                             *
 * ------------------------------------------------------------------------ */

VOID READJOYSTICK(VOID)
{
  UNSIGNED INT I,INDEX;
  UNSIGNED CHAR *CTRLPORT=(UNSIGNED CHAR*)PIO1_CTRL;
```

```
UNSIGNED CHAR *DATAPORT=(UNSIGNED CHAR*)PIO1_DATA;
UNSIGNED INT PIOPORT1[3];
DOUBLE  A= 0.1, XX, YY, ZZ;
 *CTRLPORT=0X9B;   /* SET ALL PORTS (A,B,C) INTO INPUT MODE (READ ONLY) */
 INDEX=10;         /* POSITION FOR X-DIGITS IN STRING JOYSTICK   */
FOR (I=0;I<3;I++)
 PIOPORT1[I] = *(DATAPORT+I);
XX = (DOUBLE)PIOPORT1[0]-128.0;
YY = (DOUBLE)PIOPORT1[1]-128.0;
IF (XX >= 0.0)
 XX =  XX*XX/100;
ELSE
 XX = -XX*XX/100;
IF (YY >= 0.0)
 YY =  YY*YY/100;
ELSE
 YY = -YY*YY/100;
JOYSTICK.X = A*(XX) + (1.0-A)*JOYSTICK.X;
JOYSTICK.Y = A*(YY) + (1.0-A)*JOYSTICK.Y;
```

```
IF (PIOPORT1[2]==0X03)
  SETVME((UNSIGNED CHAR *)VME9210,0X00);  /* NO BUTTON PRESSED    */
ELSE {
       SETVME((UNSIGNED CHAR *)VME9210,0X02);  /* IF ANY BUTTON PRESSED */
 }
}


DOUBLE INSENSITIVE(DOUBLE Z)
{
 IF (Z >= 10.0)
  RETURN (Z - 10.0);
 ELSE
  IF (Z <= -10.0)
   RETURN (Z + 10.0);
  ELSE
   RETURN 0.0;
}

VOID DISPLAYJOYSTICK()
{
 CONVERTINT(BCDSTRING+9, (INT)JOYSTICK.X);
```

```
BCDSTRING[3]='0';
BCDSTRING[4]='3';
BCDSTRING[6]='4';
BCDSTRING[7]='0';
SIOOUT(0,BCDSTRING);

CONVERTINT(BCDSTRING+9, (INT)JOYSTICK.OMEGA);
BCDSTRING[3]='0';
BCDSTRING[4]='4';
BCDSTRING[6]='4';
BCDSTRING[7]='0';
SIOOUT(0,BCDSTRING);     /* OUTPUT UPDATED POSITION STRING TO SCREEN */
RETURN;
}


/****************************************************************************
END OF UTILS.C
****************************************************************************/
```

APPENDIX I: SOURCE CODE (SERIAL.C)

```
/* ------------------------------------------------------------------------ *
 *                                                                          *
 * FILE:       S E R I A L . C                                              *
 *                                                                          *
 * ENVIRONMENT:  GCC COMPILER V2.7.2                                        *
 * LAST UPDATE:  26 FEBRUARY 1997                                           *
 * NAME:       THORSTEN LEONARDY                                            *
 * PURPOSE:     PROVIDES ROUTINES FOR SERIAL INPUT
AND OUTPUT TO THE 68C681                *
 *              ON THE TAURUS BOARD.                                        *
 *                                                                          *
 * COMPILED:    >GCC -C -M68040 -O SERIAL.O SERIAL.C*
 *                                                                          *
 * ------------------------------------------------------------------------ */

#INCLUDE "SHEPHERD.H"
#INCLUDE "SERIAL.H"

/* ------------------------------ *
 * GLOBAL VARIABLES               *
 * ------------------------------ */

```

```
/* UNSIGNED INT COUNTER;      /* COUNT THE INTERRUPTS          */
UNSIGNED CHAR  INPORTA;       /* CHARACTER READ FROM SERIAL PORT */

/* VT100 CONTROL SEQUENCES */

/* POSITION CURSOR, CUP = ESC [ '0' '0' ; '0' '0' H */
UNSIGNED CHAR VT100XY[9]={27,91,48,48,59,48,48,72,0};   /* POSITION CURSOR */

/* ERASE IN DISPLAY ED TO CLEAR THE SCREEN */
UNSIGNED CHAR CLRSCR[5]={27,91,50,74,0};      /* ESC [ '2' J */

/* ESC-SEQUENCE EL (ERASE IN LINE) TO ERASE A LINE */
UNSIGNED CHAR CLRLINE[6]= {5,27,91,50,75,0};  /* ESC [ '2' K */

/* ESC-SEQUENCE PRINT SCREEN (ESC [ I) */
UNSIGNED CHAR PRTSCR[4]= {27,91,105,0};      /* ESC [ I */


/* ESC-SEQUENCE SGR (SELECT GRAFIK RENDITION) (ESC [ 0 M ) */
UNSIGNED CHAR CURSOROFF[5]= {27,91,0,109,0};    /* CURSOR BLINK OFF */

/* ---------------------------------------------------------------------- *
```

```
* SIOOUT()                                                  *
* ENVIRONMENT:  GCC COMPILER V2.7.2                          *
* LAST UPDATE:  07 JANUARY 1997                              *
* NAME:         THORSTEN LEONARDY                            *
* PURPOSE:      THIS FUNCTION OUTPUTS A STRING
TO ONE OF THE TWO SERIAL                                     *
*               PORTS.                                       *
*                                                            *
* HOSTFLAG      0 -> OUTPUTS TO CONSOLE (PORT A) *
*               1 -> OUTPUTS TO HOST    (PORT B)             *
*                                                            *
* S             POINTER TO THE OUTPUT STRING                 *
*                                                            *
* ----------------------------------------------------------- */

VOID SIOOUT(INT HOSTFLAG, UNSIGNED CHAR *S)
{
  UNSIGNED CHAR *P=(UNSIGNED CHAR *)CONSOLE;

  IF (HOSTFLAG) P+=8;        /* ACCESS HOST REGISTERS    */
                             /* OTHERWISE ACCESS CONSOLE */
  WHILE(*S) {
    WHILE ((*(P+1)&4)==0);  /* SRA: WAIT UNTIL TX READY */
    *(P+3)=*S++;            /* OUTPUT CHARACTER         */
```

```
  }

  RETURN;
} /* END OF SIOOUT */



/* ---------------------------------------------------------------------------- *
 * GOTOXY()                                                                  *
 * ENVIRONMENT:  GCC COMPILER V2.7.2                                         *
 * LAST UPDATE:  14 FEBRUARY 1997                                            *
 * NAME:         THORSTEN LEONARDY                                           *
 * PURPOSE:        THIS FUNCTION POSITIONS THE CURSOR ON THE SCREEN.        *
 *                                                                           *
 * X          ROW FOR CURSOR POSITION (X=0..20)                             *
 * Y          COLUMN FOR CURSOR POSITION (Y=1..80)                          *
 *                                                                           *
 * ---------------------------------------------------------------------------- */

VOID GOTOXY(INT X, INT Y)
{

  IF ((X>0)&(X<81)&(Y>0)&(Y<33)) {
```

```
      VT100XY[2]=48+X/10;
      VT100XY[3]=48+X%10;
      VT100XY[5]=48+Y/10;
      VT100XY[6]=48+Y%10;
      SIOOUT(0,VT100XY);    /* OUTPUT ESCAPE -SEQUENCE */
    }
    RETURN;
}


/* ------------------------------------------------------------------------ *
 * SIOINIT()                                                                 *
 * ENVIRONMENT:  GCC COMPILER V2.7.2                                        *
 * LAST UPDATE:  26 FEBRUARY 1997                                           *
 * NAME:         THORSTEN LEONARDY                                          *
 * PURPOSE:      THIS FUNCTION INITIALIZES BOTH
         SERIAL PORTS. IN ADDITION,                                          *
 *               PORT A (CONSLE)IS INITIALIZED FOR
                 INTERRUPT DRIVEN I/O                                        *
 * ------------------------------------------------------------------------ */

VOID SIOINIT(VOID)
{
```

```
UNSIGNED CHAR *P=(UNSIGNED CHAR*)CONSOLE;/* BASE ADDRESS FOR 68C681 DUART */
LONG *VADR;                    /* FOR VBA REGISTER ENTRY      */
/* ------------------------------------------------------------------ */
/* INITIALIZE CONSOLE (PORT A)                                         */
/* ------------------------------------------------------------------ */
/* ATTENTION: THESE SETTINGS HAVE TO AGREE WITH THE SETTINGS FOR      */
/* YOUR TERMINAL (I.E. LAPTOP COMPUTER)                                */
*(P+2)=(UNSIGNED CHAR)0X2A;     /* CRA: RESET RX,DISABLE RX & TX   */
*(P+2)=(UNSIGNED CHAR)0X1A;     /* CRA: RESET MR POINTER,          */
*(P+0)=(UNSIGNED CHAR)0X13;     /* MR1A: RX CONTROLS RTS,          */
                                /*    8 BITS, NO PARITY            */
*(P+0)=(UNSIGNED CHAR)0X07;     /* MR2A: NORMAL MODE, 1 STOP BIT   */
*(P+1)=(UNSIGNED CHAR)0XBB;     /* SET BAUD RATE 9600 BAUD         */
*(P+2)=(UNSIGNED CHAR)0X15;     /* ENABLE RX AND TX                */
/* ------------------------------------------------------------------ */
/* INITIALIZE HOST (PORT B)                                            */
```

```
/* ------------------------------------------------------------------------ */
*(P+10)=(UNSIGNED CHAR)0X1A;        /* CRB: RESET MR POINTER */
*(P+8)=(UNSIGNED CHAR)0X13;         /* MR1B: NO PARITY, 8 BITS */
*(P+8)=(UNSIGNED CHAR)0X07;      /* MR2B: NORMAL MODE, 1 STOP BIT */
*(P+9)=(UNSIGNED CHAR)0XBB;       /* SET BAUD RATE 9600 BAUD */
*(P+10)=(UNSIGNED CHAR)0X15;       /* CRB: ENABLE RX AND TX */


/* ------------------------------------------------------------------------ */
  /* IT FOLLOWS THE INTERRUPT SPECIFIC PART FOR PORT A */
/* ------------------------------------------------------------------------ */
*(P+5)=(UNSIGNED CHAR)0X02;      /* ISR: SET INTERRUPT MASK FOR RXRDY A */
*(P+12)=0X60;                /* IVR: PLACE INTERRUPT VECTOR      */
                             /* 0X60 ACCESSES VBA AT BASE+0X180   */
VADR=(LONG*)0XFFE40180;         /* VBA ADDRESS FOR INTHANDLER */
*VADR=(LONG)INPORTAHANDLER;     /* WRITE ADDRESS INTO VBR */

P=(UNSIGNED CHAR*)ISM_SERIAL;       /* ISM CONFIGURATION REGISTER FOR SIO */
*P=0X09;                  /* INTERRUPTS TO 68040 ON LIRQ-1      */
```

```
P=(UNSIGNED CHAR*)VIC_LIRQ1;     /* VIC068 LICR FOR LIRQ-1 FROM
ISM   */
*P=0X01;                    /* ASSERT IRQ-1 FROM VIC TO 68040     */


RETURN;
}




/*************************************************************************
 ASSEMBLER ROUTINES
*************************************************************************/

/* ---------------------------------------------------------------------- *
* INPORTAHANDLER()                                  *
*                                    *
* ENVIRONMENT:  GCC COMPILER V2.7.2                          *
* LAST UPDATE:  27 JANUARY 1997                        *
* NAME:       THORSTEN LEONARDY                         *
 * PURPOSE:       INTERRUPT HANDLING ROUTINE FOR INTERRUPTS
FROM 68C681 DUART *
```

```
 *          IT READS A CHARACTER INPUT FROM THE KEYBOARD INTO VARIABLE *
 *          INPORTA, INCREMENTS A COUNTER, AND OUTPUTS THE CHARACTER *
 *          TO THE SCREEN. IF A CR IS TYPED AT THE KEYBOARD, AN *
 *          ADDITIONAL LINEFEED (0X0A) IS ADDED TO THE <CR> (0X0D). *
 *                                         *
 * ------------------------------------------------------------------------ */

ASM("

    .EVEN
    .TEXT
    .GLOBL _INPORTAHANDLER

_INPORTAHANDLER:

    LINK    A6,#-128        /* ALLOCATE 184 BYTES ON STACK TO ... */
    FSAVE   A6@(-128)
    MOVEML  D0-D7/A0-A5,SP@-  /* SAVE REGISTERS (14*4 BYTE) */

    MOVE.L #0XFFF4A000,A2     /* BASE ADDRESS OF 68C681 DUART */
    MOVE.B 3(A2),D2           /* RHR_A: READ CHARACTER ...       */
```

```
        MOVE.B D2,_INPORTA        /* ... AND COPY TO INPORTA          */

        MOVEML   SP@+,D0-D7/A0-A5
        FRESTORE A6@(-128)
        UNLK     A6


        RTE
");



/***************************************************************************
 END OF SERIAL.C
***************************************************************************/
```

APPENDIX J: SOURCE CODE (CONSOLIDATED HEADER FILES)

The following code was modified by: Professor Kanayama, Thorsten Leonardy, Edward Mays, and Ferdinand A. Reid.

```
/* shepherd.h */

#ifndef SHEPHERD_H
#define SHEPHERD_H

/* ------------------------------------------------ *
 * Base Addressees for accessing Servo Control Cards *
 * ------------------------------------------------ */

#define VME9210    0xffff0400    /* Base Address analog out to servo    */
#define VME9421    0xffff0000    /* Base address data in from servo     */
#define VME2170    0xffffff00    /* Base address data out to servo      */
#define VMECTR1    0xffff6000    /* VME Counter for driving motor       */
#define VMECTR2    0xffff6100    /* VME Counter for steering motor      */


/* ------------------------------------- *
 * defines for general Interrupt Handling *
```

```
* ------------------------------------- */

#define VIC_LIRQ1  0xfff44027          /* VIC068 Register for LIRQ-1 */
#define VIC_LIRQ2  0xfff4402b          /* VIC068 Register for LIRQ-2 */
#define VIC_LIRQ3  0xfff4402f          /* VIC068 Register for LIRQ-3 */
#define VIC_LIRQ4  0xfff44023          /* VIC068 Register for LIRQ-4 */
#define VIC_LIRQ5  0xfff44037          /* VIC068 Register for LIRQ-5 */
#define VIC_LIRQ6  0xfff4403b          /* VIC068 Register for LIRQ-6 */
#define VIC_LIRQ7  0xfff4403f          /* VIC068 Register for LIRQ-7 */

#define VIC_TTR    0xfff44043          /* Transfer Timeout Register */
                                  /* see p. 4-2 TAURUS Manual  */
#define VIC_ICR    0xfff440af          /* VIC Interface Configuration */


#define enable()   asm("move.w #0x2000,sr")  /* enable interrupts */
#define disable()  asm("move.w #0x2700,sr")  /* disable interrupts */


/* defines for Vector base register entries */
#define VBA_TIMER  0xffe40130     /* Vector table address for Timer-5 ISR */
#define VBA_PI     0xffe40118     /* Vector table entry for IP interrupt */
```

```
/* ---------------------------------------- *
 * defines for interrupt steering mechanism *
 * ---------------------------------------- */

#define ISM_TIMER  0xfff48004    /* ISM Configuration Register for Timer A */
#define ISM_PI     0xfff48008    /* ISM Configuration Register for PI      */
#define ISM_SERIAL 0xfff48001    /* ISM Configuration Register for serial IO  */

#define APP_ICR    0xfff4800c    /* abort/processor interrupt control register */



/* ------------------------------------------------- *
 * Base Addressees for accessing Parallel IO-Ports   *
 * ------------------------------------------------- */
#define PIO1_CTRL  0xfff40003          /* control register for PIO-1 */
#define PIO1_DATA  0xfff40000          /* data register for PIO-1 Port A */
#define PIO2_CTRL  0xfff40007          /* control register for PIO-2 */
#define PIO2_DATA  0xfff40004          /* data register for PIO-2 Port A */

```

```
/* ------------------------------------------------ *
 * Base Addressees for 68030 Input/Output Program    *
 * as outlined in Taurus Manual, Chapter 6           *
 * ------------------------------------------------ */
#define IOP_CMDBLK     0xffe00000        /* address for IOP Command Block */
#define IOP_START      0x01              /* command to start IOP */
#define IOP_STOP       0x00              /* command to stop IOP */
#define IOP_COMPLETE   0x80              /* mask for operation complete */

#define IOPB_CONFIGURE 0xe0              /* command to configure IOBP */
#define IOPB_UNIT_OMNI0 0x10             /* unit # for omnimodule #0 */
/* ---------------------------------------------------------- *
 * definitions for 68030 Input/Output Program, (Leo, 05/13/97) *
 * ---------------------------------------------------------- */


/* Input/Output Parameter Block structure, according Taurus Manual, p. 6-4 */
typedef struct {
   unsigned char cmd;   /* command */
   unsigned char error; /* error status */
   unsigned short options;   /* options */
```

```
    unsigned short reserved; /* reserved, do not use */
    unsigned char unit;       /* unit number */
    unsigned char destUnit;   /* destination unit */
    unsigned long blockNumber;/* logical Block number */
    unsigned long txCount;    /* Transfer count, # of bytes to transfer */
    unsigned long *ptrSrc;    /* address of source */
    unsigned long *ptrDst;    /* Address of destination */
}IOPB;

/* Command Block structure according to Taurus Manual, p. 6-3 */
typedef struct {
    unsigned char cmd;          /* status and command register */
    unsigned char reserved[3];  /* not yet used */
    IOPB *ptrToIOPB;    /* pointer to IOBP */
}CMD_BLOCK;


/* Omnimodule support  block structure according to Taurus Manual p. 6-12 */
typedef struct {
    unsigned long options;     /* 4 bytes options, unused            */
    unsigned long *ptrInit;    /* pointer to initialization routine   */
    unsigned long *ptrTask;    /* pointer to task                     */
```

```
  unsigned long *ptrIntr;   /* pointer to interrupt servicing routine */
}OSB;

IOPB  iopbOMNI0; /* IOBP for Omnimodule 0 (used for serial I/O to VT100 */
OSB   osbOMNI0;  /*  OSB for Omnimodule 0 (used for serial I/O to VT100 */

/* ------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------ */

unsigned int intCounter, testCounter;   /* count the interrupts          */
unsigned int demo;          /* switch to run demo see driver() in movement.c */
unsigned short timer_in_ms;  /* desired timer period in ms */


/* ----------------------------------------------------------- *
 * definitions for inertial measurement routines (imu.c)        *
 * ----------------------------------------------------------- */

/* added 10 Sep 97 */
typedef struct {
 unsigned short ax;  /* linear acceleration in x-direction */
 unsigned short ay;  /* linear acceleration in y-direction */
```

```
unsigned short az;  /* linear acceleration in z-direction */
unsigned short omega_z; /* angular velocity in z-direction */
}IMU;

IMU imu;            /* stores most recent IMU data (updated with */
                    /* every 10ms timer interrupt            */

/* -------------------------------------------------------- *
 * definitions for Joystick Control, (Leo, 05/10/97)         *
 * -------------------------------------------------------- */

typedef struct {
  double x;        /* x position (or velocity) */
  double y;        /* y position (or velocity) */
  double omega;     /* angular velocity */
  unsigned char state; /* status of parallel port 1, channel C */
}JPOINT;
JPOINT joyStick; /*global*/

typedef struct {
  double x;
  double y;
```

```
}point;


typedef struct {
  point  coord;
  double heading;
  double kappa;
}Configuration;
Configuration vehicle; /*global*/

typedef struct {
  double Speed;
  double Theta;
  double Omega;
}vehicleMotion;
vehicleMotion motion,motion0; /*global*/

typedef struct {
  double rho;
  double alpha;
}polar;

```

```

/* ---------------------------- */
/* definitions for wheel control */
/* ---------------------------- */

/* write these masks to VME2710 at address 0xffffff00 in order to make     */
/* the specific motor drive! May wish to logical OR with previous settings */

#define TURN_FR    0x00004000    /* turn wheel 1 (front right) */
#define TURN_FL    0x00020000    /* turn wheel 2 (front left)  */
#define TURN_RR    0x00100000    /* turn wheel 3 (rear right)  */
#define TURN_RL    0x00800000    /* turn wheel 4 (rear left)   */

#define DRIVE_FR   0x00000004    /* drive wheel 1 (front right) */
#define DRIVE_FL   0x00000020    /* drive wheel 2 (front left)  */
#define DRIVE_RR   0x00000100    /* drive wheel 3 (rear right)  */
#define DRIVE_RL   0x00000800    /* drive wheel 4 (rear left)   */

#define ALL_WHEELS 0x00924924    /* select all wheels for turning */
                        /* and driving */

/* -------------------- *
```

```
 * function definitions *
 * -------------------- */

void setVME(unsigned char *board, unsigned char data);
void toggleVME(unsigned char *board, unsigned char data);
void initBoards(void);
void piTest(void);
void piHandler(void);
void advanceCount();
 /* global variable to make joystock coordinates accessible */

#define ARRAY_SIZE      4
#define DegreesToRads   0.0174532925
#define RadsToDegrees   57.29577951308232
#define DeltaT          0.01


double desiredAngleRates[ARRAY_SIZE],
       desiredAngleRates0[ARRAY_SIZE],
       desiredSpeeds_F[ARRAY_SIZE],
       desiredAngleRates_F[ARRAY_SIZE],
       desiredSpeeds[ARRAY_SIZE],
```

```
        actualSpeeds[ARRAY_SIZE],    /* 28 May ejm */
        actualAngleRates[ARRAY_SIZE],
        DigitToCmDrive[ARRAY_SIZE],
        Display_Speeds[ARRAY_SIZE],
        Display_Steers[ARRAY_SIZE],
        desiredAngles[ARRAY_SIZE],
        desiredAngles0[ARRAY_SIZE],
        actualAngles[ARRAY_SIZE];
short   Steer_Digits[ARRAY_SIZE],
        Speed_Digits[ARRAY_SIZE];
double            WheelDriveAct[ARRAY_SIZE],
                  WheelDriveDes[ARRAY_SIZE];
unsigned long int WheelDriveAct0[ARRAY_SIZE],
                  WheelDriveAct1[ARRAY_SIZE],
                  driveReadings[ARRAY_SIZE];
double            WheelDirAct[ARRAY_SIZE],
                  WheelDirDes[ARRAY_SIZE],
```

```
                PreviousCountSpeed[ARRAY_SIZE],
                PreviousCountSteer[ARRAY_SIZE];
unsigned long int    WheelDirAct0[ARRAY_SIZE],
                WheelDirAct1[ARRAY_SIZE],
                steerReadings[ARRAY_SIZE];
int mode,
    oldMode,
    mode0state,
    mode5state,
    modeTstate,
    Flag,
    oldFlag,
    edCounter,
    hallSensor3;
unsigned int intCounter, testCounter;    /* count the interrupts            */
/*unsigned long int */
double previousCount,previousCountSteer, Omega_Speed,
```

```
previousCountSpeed;/*previousCount represents infinity */

double K1[ARRAY_SIZE],
       K2[ARRAY_SIZE],
       K3[ARRAY_SIZE],
       K4[ARRAY_SIZE],
       K6[ARRAY_SIZE];    /* slope based on input units vs output velocity,   */
                 /* input range from 0- 1020, feedback constant      */
                 /* K3 is the inverse of (86.975velocity/1020 digit) */


#endif

/*********************************************************************
 End of shepherd.h
 *********************************************************************/


#ifndef __MOVEMENT_H__
#define __MOVEMENT_H__

#include "shepherd.h"
```

```
#define PI          3.14159265358979323846
#define DPI         6.28318530717958647692  /* PI*2     */
#define HPI         1.570796327             /* PI/2     */
#define QPI         0.785398163             /* PI/4     */
#define QPIby500    0.0015707963
                    /* QPI/(5 seconds/deltaT) */




double wheel_speed[4], wheel_angle[4];

void initMovement();
void setupPolar(polar []);
void wheelMotion();
void bodyMotion();
void driver();
void joystickMotionInterface(void);
```

```
extern double desiredAngleRates[],
        desiredSpeeds[],
        PreviousCountSpeed[],
        PreviousCountSpeed[],
        PreviousCountSteer[];
polar whp[4];
double pathLength,thetaDot,omegaDot,speedDot;
/* *************Items for tangential motion ************** */
double sigma;
double radius;
double ai[4], bi[4];
typedef struct {
 Configuration   config;
 point        center;
 double        radius;
```

```
double      a;
double      b;
double      c;
} LINE;

static LINE  currentPath; /* holds the current path element values */
Configuration incrementalMotion, holdVehicle;
static double deltaS;

void tangentialMotion();
void circularArc(double length, double alpha);
void defineConfig(double x,double y,double theta,double kappa);
void compose();
double steer();
void constants();
double Psi(point p1,point p2);
double distance(point p1,point p2);
void initTangent();

#endif


```

```
#ifndef __MOTOR_H__
#define __MOTOR_H__

#include "shepherd.h"

/* ------------------------------------------------ *
 * Base Addressees for accessing Servo Control Cards *
 * Used in Home Testing
 * ------------------------------------------------ */
#define   SteerDriveInteract  .02    /* used to give stability to wheel      */

#define   RadRateTodigit  195.3789   /* digit/radpersec*/

#define   digitToRadDrive  -6.015495746e-5
          /* driving constant rad/count = DPI/104450    May 8  */
          /* Experimental Results by Ed Mays            May 7  */
          /* Wheel 1 count = 104456                      */
          /* Wheel 2 count = 104435                      */
          /* Wheel 3 count = 104454                      */
          /* Wheel 4 count = 104455                      */
```

```
            /* Average count = 104450                     */
            /* cf. 2048 * 51 = 104448                     */
#define digitToCmDrive  0.0011369287
            /* driving constant cm/count = digitToRadDrive*18.9cm  5/8/97 */

#define digitToRadSteer -6.817692391e-5
            /* steering constant rad/count = DPI/(2048*45)  19 Apr  */

#define SteerFBGain 0.000;      /* steering feedback gain          */
#define DriveFBGain 0.000;      /* driving  feedback gain          */
#define DigitsHigh  1023
#define DigitsLow  -1024
#define WheelRadius 18.9        /* prev def in cm */
#define VME9210    0xffff0400    /* Base Address analog out to servo    */
#define VME9421    0xffff0000    /* Base address data in from servo    */
#define VME2170    0xffffff00    /* Base address data out to servo     */
#define VMECTR1    0xffff6000    /* Counter                      */
#define K5        87.4          /* control feedback constant (cm/sec) variable 28 May ejm */
#define DriveFeedBackGain  0.8   /*.8 control drive feedback gain 28 May ejm */
#define angularK3       0.96963 /* digit/rotational speed (rad/sec)   */
#define steerFeedbackGain  100.0 /* steering Feedback gain  */
#define angularK5       5.23598
```

```
#define angleFeedbackGain  1000.0

extern unsigned char clrLine[6];  /* ESC-Sequence for clear line          */
extern char bwheeldrivecdString[];/* defined in shepherd.c               */
extern unsigned char bcdString[];

double Drive_Feedback[ARRAY_SIZE];

extern double desiredAngleRates[],
            desiredSpeeds[],
            PreviousCountSpeed[],
            PreviousCountSteer [],
            DigitToCmDrive[],
            Display_Speeds[],
            Display_Steers[],
            desiredAngles[],
            actualAngles[];

/* -------------------- *
 * function definitions *
 * -------------------- */
void driveSpeed(short []);
```

```
void driveSteer(short []);
void driveMotors();
void wheelDrive(void);
void allStop(void);
void updateEncoders(void);
void updateWheelDrive(void);
void updateWheelSteer(void);
void displayDirections(void);
void displaySpeed(void);
int readSteerEncoders(unsigned long int []);
void testDrive(void);
void readEncoders(void);
void accumulatedriveSpeed();
void displayDriveAngle();                /* added 15 may */
void drivingFeedback();                  /* 28 May ejm   */
double velocityReferenceTable(double,int);        /* 28 May ejm   */
void steeringFeedback();                 /*  4 June     */
double rateReferenceTable(double);            /*  4 June     */
void computeActualRates();                 /*  5 June     */
int convertDifference(int);              /*  11 June ejm */

void alignWheels(void);
```

```
void clearShaftEncoder(unsigned short motors);
void readWheelStatus(unsigned char *array);

#endif

/* Timer.h */

#ifndef TIMER_H
#define TIMER_H

/* Defines for Timer control */
#define TIMER_CTRL 0xfff41002    /* Control register for Timer A        */
#define TIMER_DATA 0xfff41000    /* Data register for Timer A           */

/* settings master mode register according to fig. 1-12               */
#define TIMER_MASTER_MODE 0xbaf0  /* timer master mode register          */
                          /* b=BCD count, 16 Bit data bus      */
                          /* 4=divide by 4                     */
                          /* 8 = Source F4 (divide by 1000)    */
```

```
                        /* 0 = don't care                */


/* settings for counter mode register according to fig. 1-17 */
#define TIMER_MODE 0x0f31       /* Counter Mode Register Bit Assignment */
                        /* 0 = no gating, count on rising edge  */
                        /* 8 = Source F4 (divide f by 1000)     */
                        /* 3 = BCD repetitive count,reload load */
                        /* 2 = count down, toggle TC            */
                /* or 1 = count down, active high Terminal Count Pulse */

/* -------------------- *
 * function definitions *
 * -------------------- */

void TimerHandler(void);
void timerStart(void);



#endif

```

```
/***************************************************************************
End of timer.h
***************************************************************************/




#ifndef __MATH_H__
#define __MATH_H__



double cos(double x);
double sin(double x);
double atan2(double x, double y);
double atan(double x);
double new_sqrt(double x);
double new_abs(double x);
double norm(double angle);
double min (double, double);
```

```
double max (double, double);
#endif


/* ------------------------------------------------------------------------ *
 *                                                      *
 * File:      S E R I A L . H                                  *
 *                                                      *
 * Environment:  GCC Compiler v2.7.2                                  *
 * Last update:  13 March 1997                                  *
 * Name:        Thorsten Leonardy                                  *
 * Purpose:     Header File for 'serial.c'                                  *
 * ------------------------------------------------------------------------ */

#ifndef __SERIAL_H__
#define __SERIAL_H__


#define CONSOLE    0xfff4a000    /* Base address 68C681 DUART */

/* -------------------- *
 * function definitions *
```

```
* ------------------- */
void inPortAHandler(void);              /* interrupt handler    */
void sioInit(void);                     /* initialize DUART     */
void sioOut(int hostFlag, unsigned char *s); /* Output a string      */
void gotoXY(int x, int y);              /* position cursor      */
#endif
/************************************************************************
 End of serial.h
 ************************************************************************/
/* ---------------------------------------------------------------------- *
 *                                                                        *
 * File:       U T I L S . H                                              *
 *                                                                        *
 * Environment:  GCC Compiler v2.7.2                                      *
 * Last update:  13 March 1997                                            *
 * Name:         Thorsten Leonardy                                        *
 * Purpose:      Header File for 'utils.c'                                *
```

```
* ------------------------------------------------------------------------ */

#ifndef __UTILS_H__
#define __UTILS_H__

/* -------------------- *
 * function definitions *
 * -------------------- */

void setVME(unsigned char *board, unsigned char data);
void toggleVME(unsigned char *board, unsigned char data);
void initBoards(void);
void piTest(void);
void piHandler(void);
/*void readClock(void); */
/*void WRITE_CLOCK(void); */

/* Modified 18 Apr  */
unsigned char b2a(int);
void b2a2(unsigned char *, unsigned char);
void convertToASCII(unsigned int ndigits, unsigned int data, char *str);
void readJoyStick(void);
```

```
double insensitive(double z);

#endif

/*****************************************************************
 End of utils.h
 *****************************************************************/
```

APPENDIX K: SHEPHERD OPERATING MANUAL

OVERVIEW

The Purpose of this document is to provide a quick guide for doing downloads for testing or other purposes. For a more detailed guide see the ***Shepherd Operators Guide (SOG)***.

The Shepherd compilation and download process is a four step process:

- ***Compile*** executable on workstation.
- ***FTP*** S-Records to laptop.
- Use Windows 95 ***HyperTerminal*** program for ***direct connection***.
- ***Run*** the program once download complete.

Compile Executable on Workstation

1. Once you have logged in on the Shepherd account, then use the ***xinit*** command to generate the X-Windows environment.

2. In the large terminal window type "***cap***" at the UNIX prompt and press the return key.

3. The alias "cap" logs you onto capella (the standard login script will scroll by). The Shepherd group uses capella (server) because of the nature of the cross compilation used for the "Taurus board" and Motorola 68040 CPU.

4. Next, in the large terminal window type "***taurus***" at the prompt and press the return key.

5. The alias "taurus" sets up the environment for compilation and print services.

6. Next, in the large terminal window type "***cd srk***" at the prompt and press the return key; this takes you to the Shepherd Real-time Kernel (~shepherd/srk) directory. While in srk you can modify or edit the require files with your favorite editor (e.g., xemacs or nedit). Once you have completed your work, save your files and compile. ***See figure1 on the next page***.

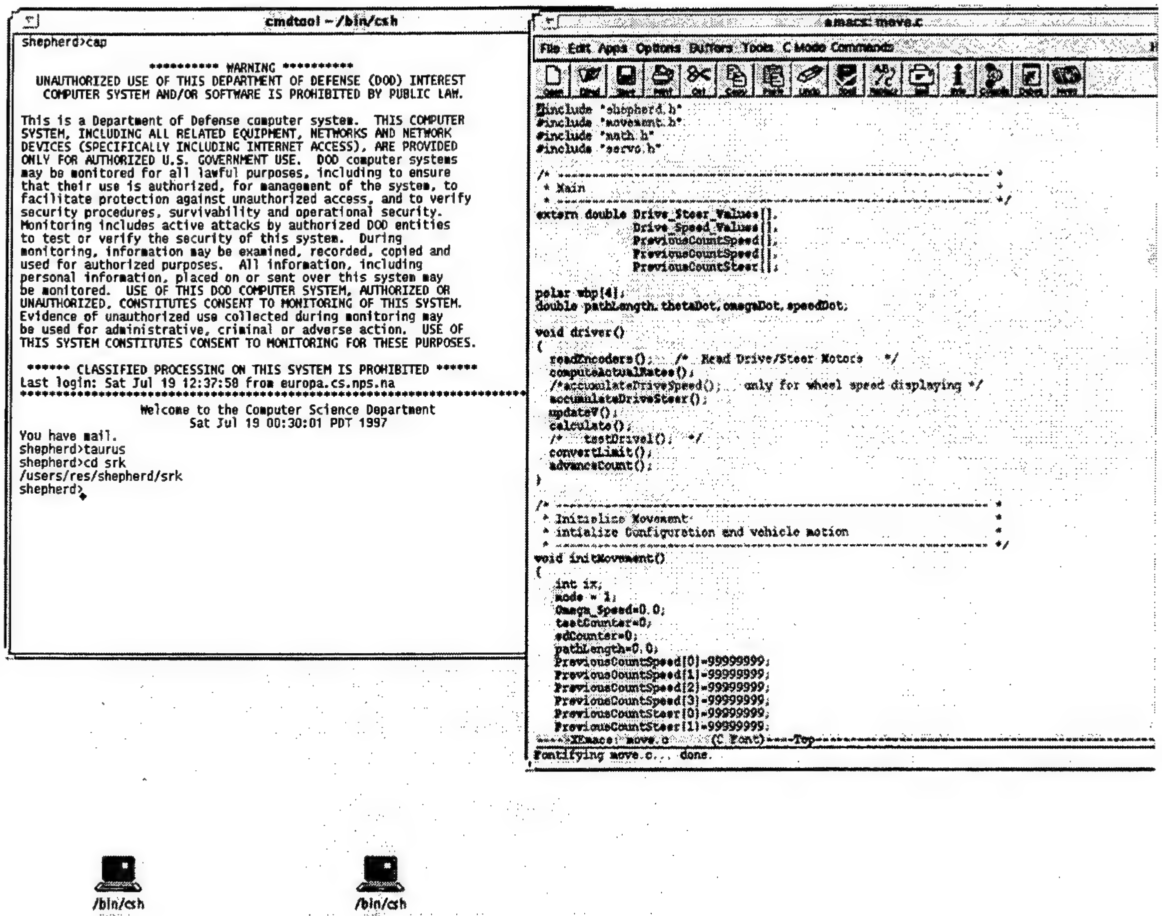

Figure 1: **The Unix Workstation Environment**

7. Compilation is done through the use of a ***makefile***. Hence, to compile all you must do is type "***make comp***" at the prompt and press the return key (this will either succeed or fail). If the compilation fails work the errors provided by the compiler and compile again (an iterative process). Once, the compilation is a success you are ready to ***FTP*** the S-records to the laptop.

FTP S-Records to Laptop

8. To begin to ***FTP*** the S-records to the laptop a few items must be accomplished. First, the robot power must be switched on (levers a, b, and c on the power supply in the "up" or closed position on the physical robot; provides power to the robot and charges the batteries).

See figure 2 (note the "up" position below represents the down position on the physical robot).

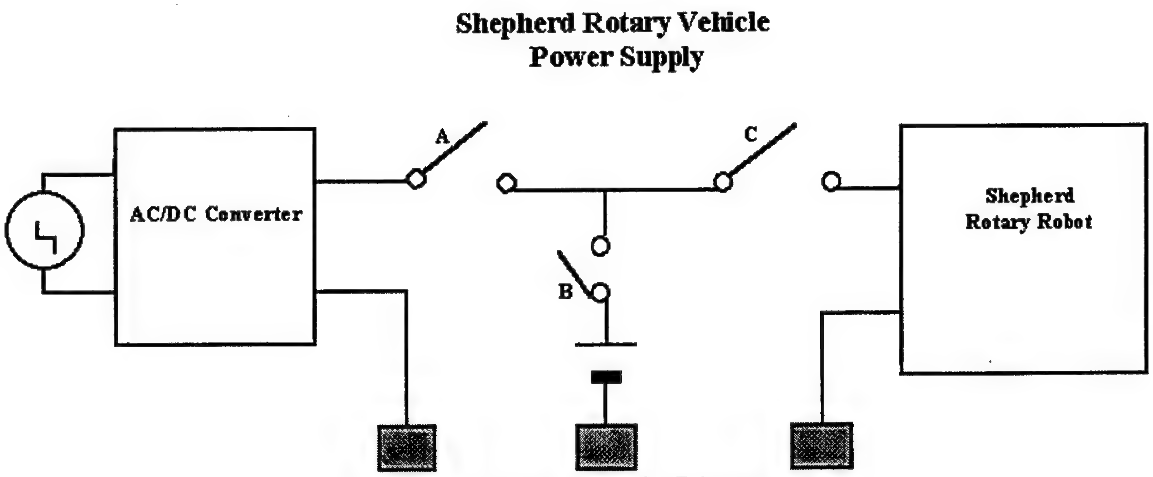

Figure 2: **Power Supply Diagram**

9. Secondly, the laptop must be on and connected to the local network via the PCMIA card (ethernet). ***Press*** the laptop "***On" button.***

10. Ensure the laptop powersupply is plugged in, and connected to the laptop.

11. Ensure the therenet cable is properly connected to the to the PC card.

12. After booting our laptop will prompt you to login as guest-- just ***"click on the cancel"*** button. You should see the Windows desk top on the laptop (figure 3 below).

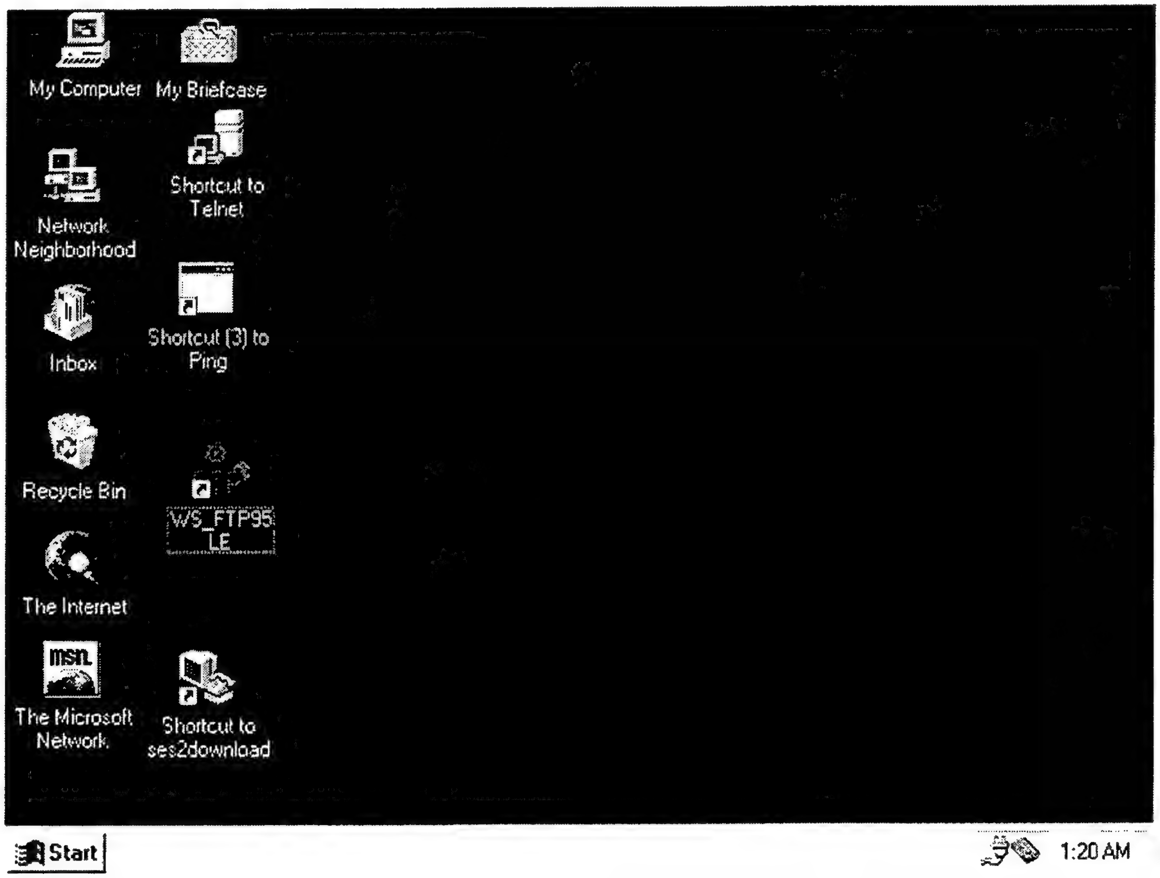

Figure 3: **Windows Desk Top**

13. The screen print below describes the way the windows should look. Now ***double click*** on the ***WS_FTP95 shortcut*** to open the ftp tool.

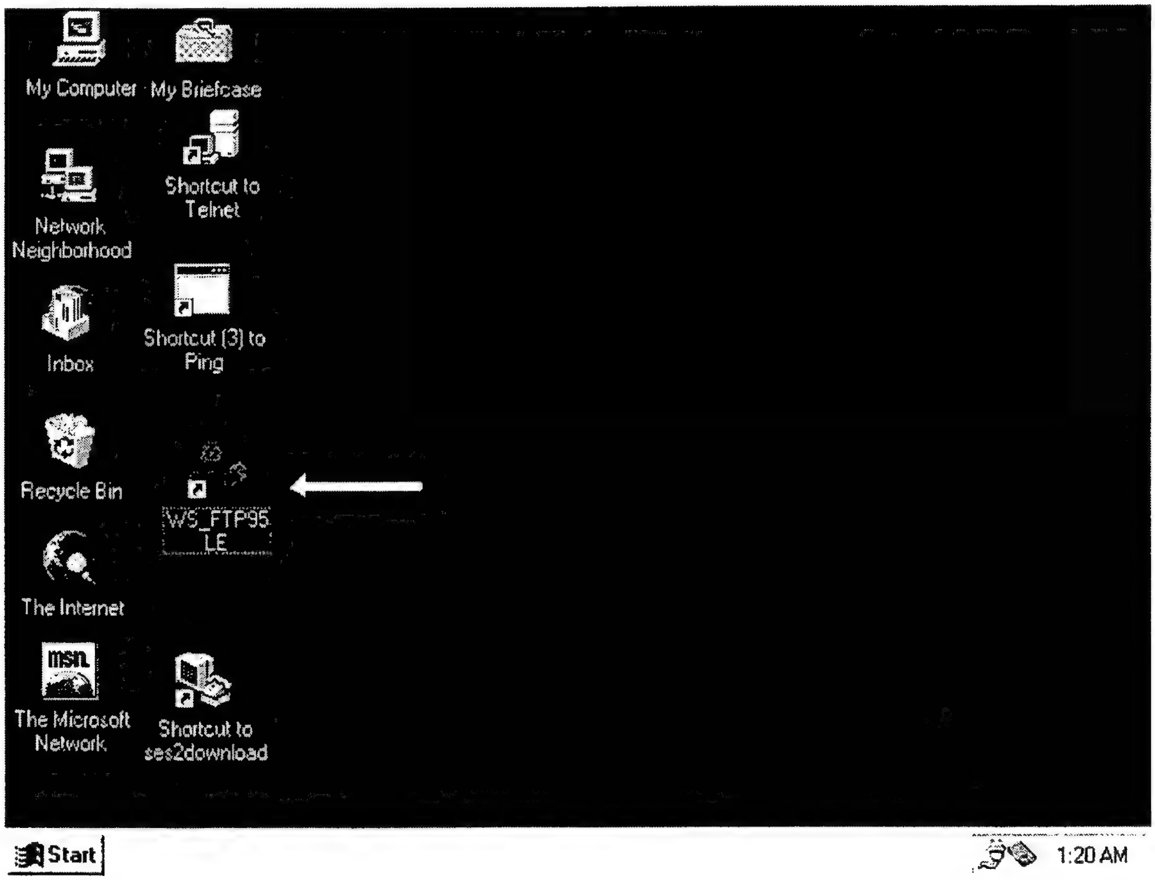

Figure 4: **Windows WS_FTP95 Shortcut**

14. At this point the ftp tool opens. ***Click*** on the ***"OK" button***. All information has been previously set for you(e.g., hostname,UserID, and Password). See figure 5.

15. The next Window has a split panel showing the Remote System (workstation) and the Local System (the laptop). The directories have been saved, so they always open to the correct directories. The file to be ftp'd is **shepherd.TXT, it** will be ftp'd from the remote system (~shepherd/srk) to the local system (c:\shepherdump). The file **shepherd.TXT** contains the S-Records that will be eventually downloaded to the actual robot CPU. To accomplish the ftp ***highlight the file*** to be transferred with your mouse and ***click on the arrow that points left*** *(See figure 6)*. The file transfer usually takes about .3 seconds.

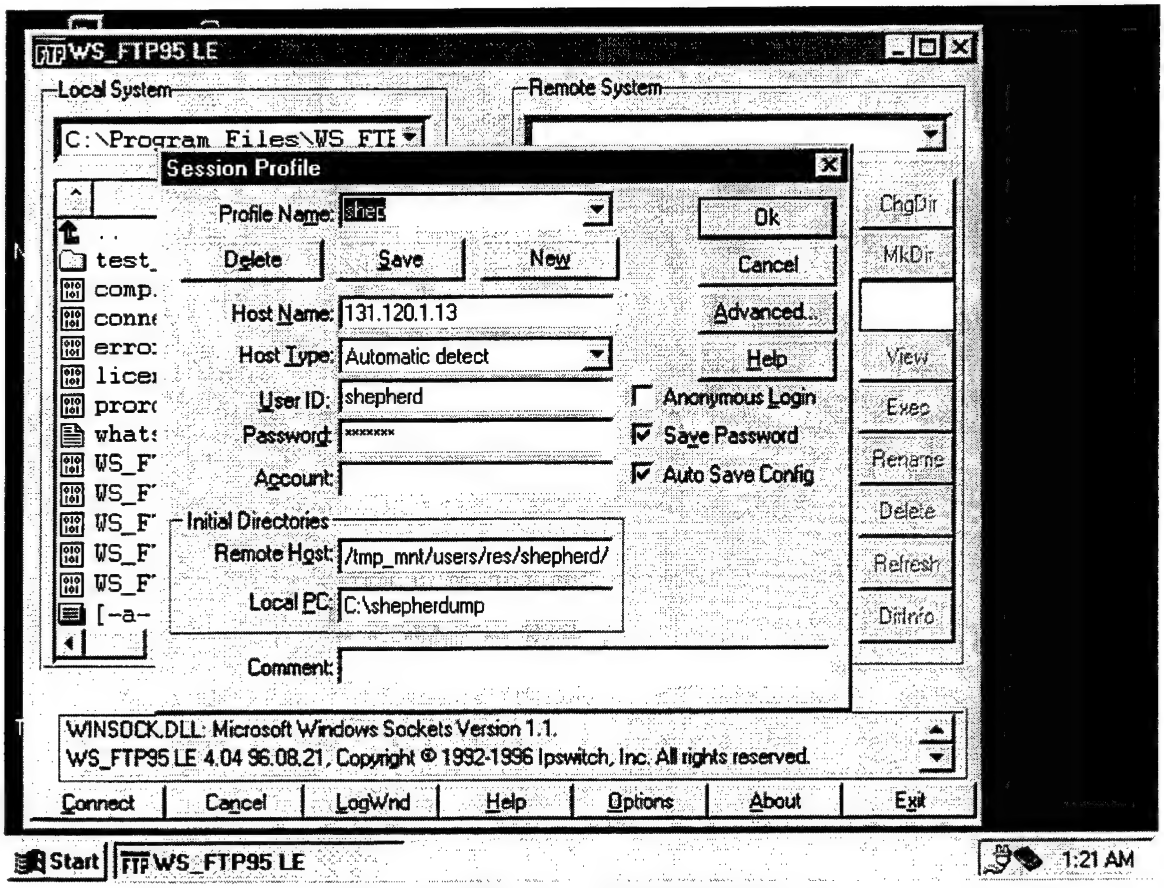

Figure 5: **Windows WS_FTP95 Tool**

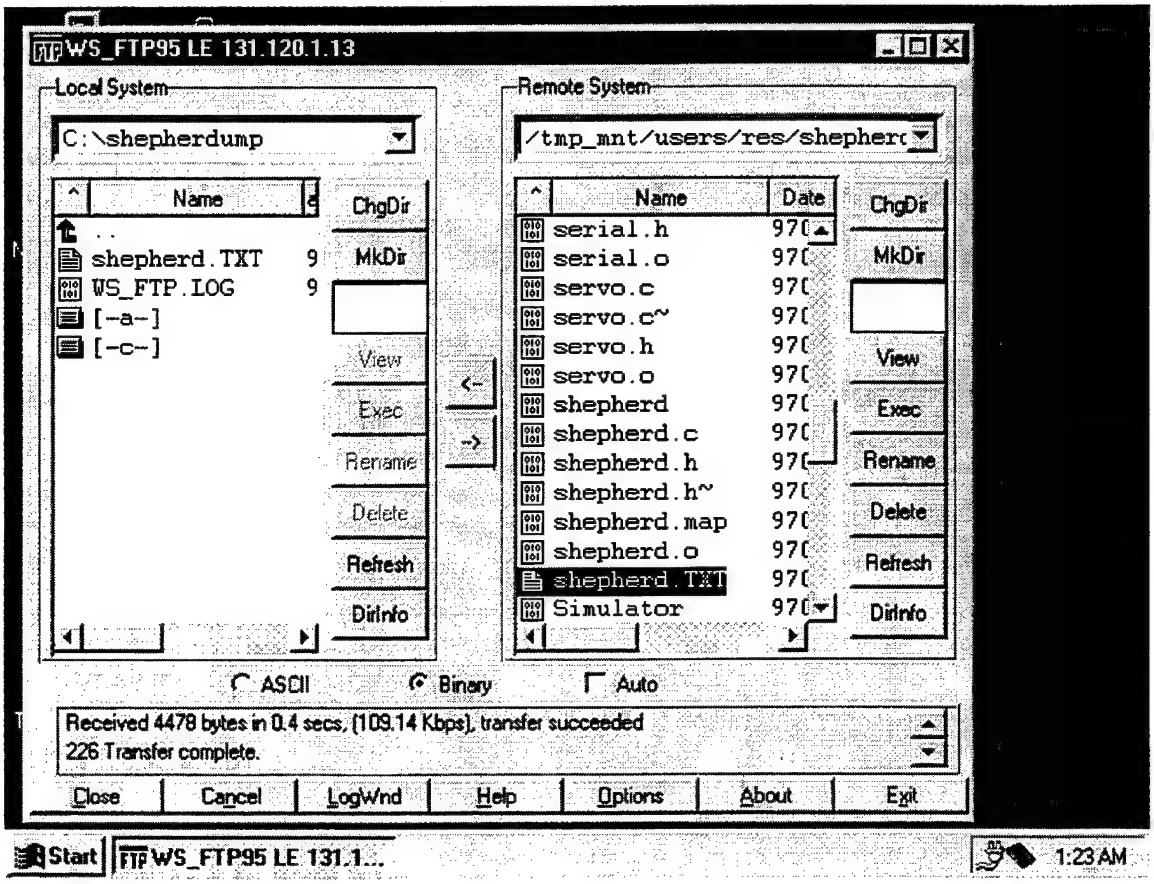

Figure 6: **WS_FTP95 Tool File Transfer**

16. The file is now on the "hard" disk of the laptop. You can now ***close the window*** or kill the process by clicking on the appropriate button (active window:upper right corner area **"X"**).

Use Windows 95 HyperTerminal Program for Direct Connection

17. You are now back at the Windows desk top. Now ***double click*** on the ***ses2download*** shortcut to open a hard-line, under Windows 95 HyperTerminal (See figure 7).

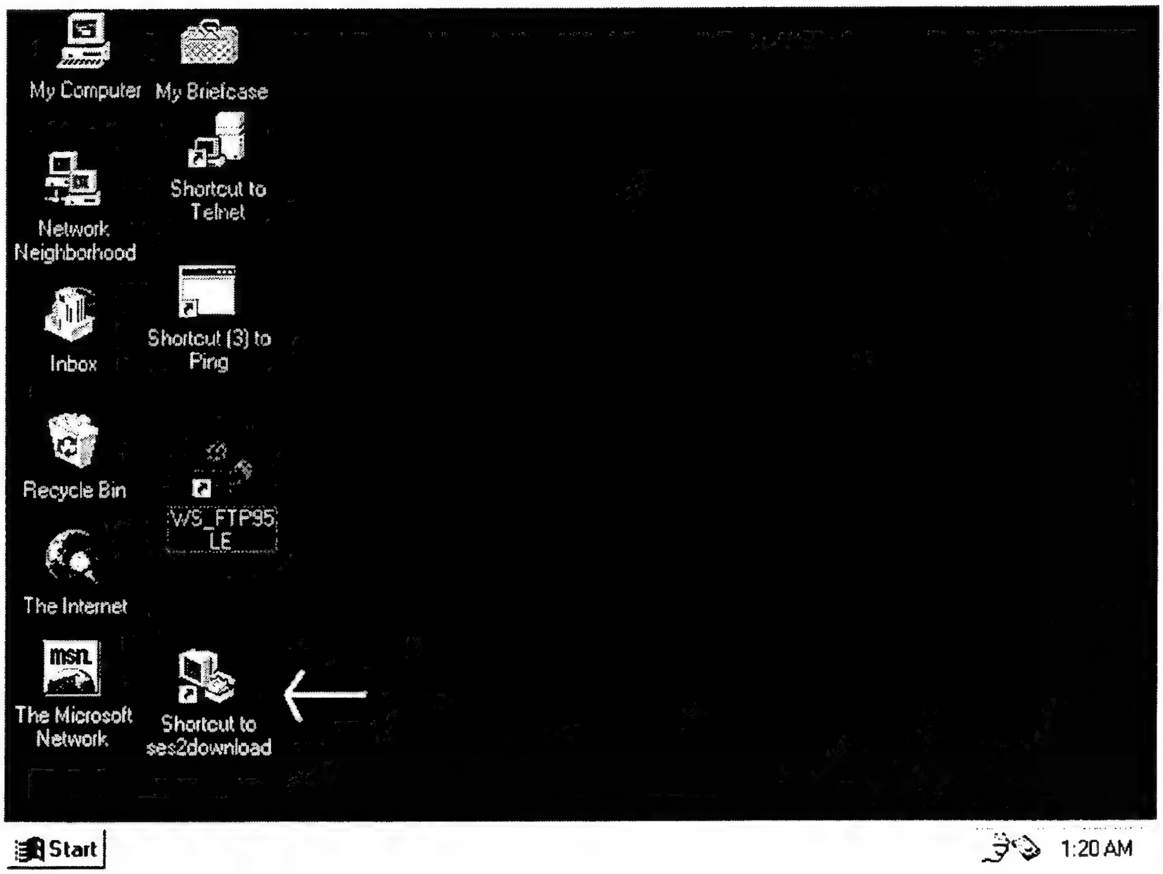

Figure 7: **Windows ses2download Shortcut**

18. The next window to appear will be the open ***HyperTerminal window*** (See Figure 8). Press the "***reset***" button on the ***OMNIBYTE, Taurus board***. The Taurus bug (debugger) prompt will appear in the ***HyperTerminal window*** (See Figure 8).

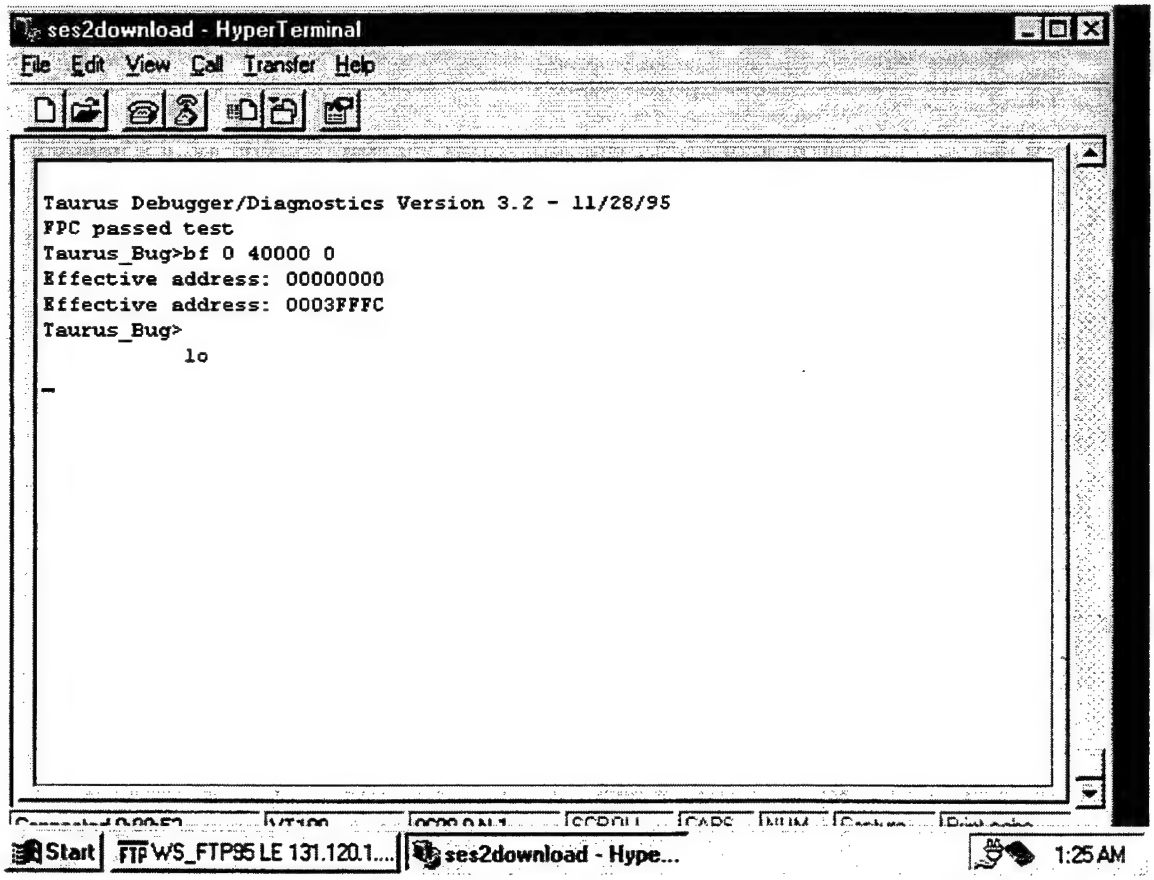

Figure 8: **Windows HypertTerminal Window**

19. Ensure the ***lever*** on the ***switch box*** is placed on ***console*** (this allows the console to emulate a VT220).

20. At the Taurus bug prompt type "***bf 0 40000 0***" and ***press the enter key***. This command is called block fill by the debugger it allows you to disable the parity error interrupt (***PEI***) and prevents problems caused by uninitialized variables. See Figure 8.

21. At the Taurus bug prompt type "***lo***" and ***press the enter key***. The "***lo***" command initiates the download from the console. See Figure 8.

22. Next place the ***lever*** on the ***switch box*** is placed on ***host*** (this makes possible the use of the RS232 protocol to download **shepherd.TXT from** c:\shepherdump to the Taurus board).

23. ***Click*** on the HyperTerminal "***Transfer "option*** and choose the "***Send Text File***". All the "***Send Text File***" parameters have been previously set, so there is no action to take in that regard.
See Figure 9.

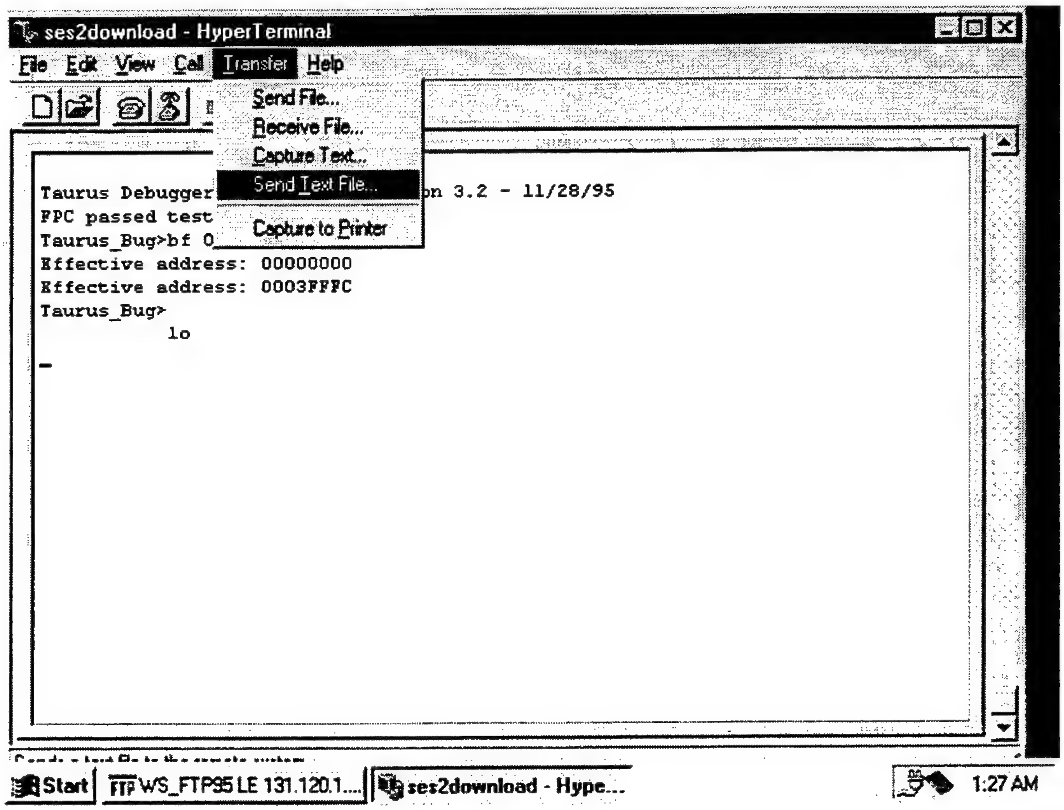

Figure 9: **Windows HypertTerminal Window**

24. Now move to the root directory and ***select the c:\shepherdump directory***, and double click on **shepherd.TXT** file. See figure 10.

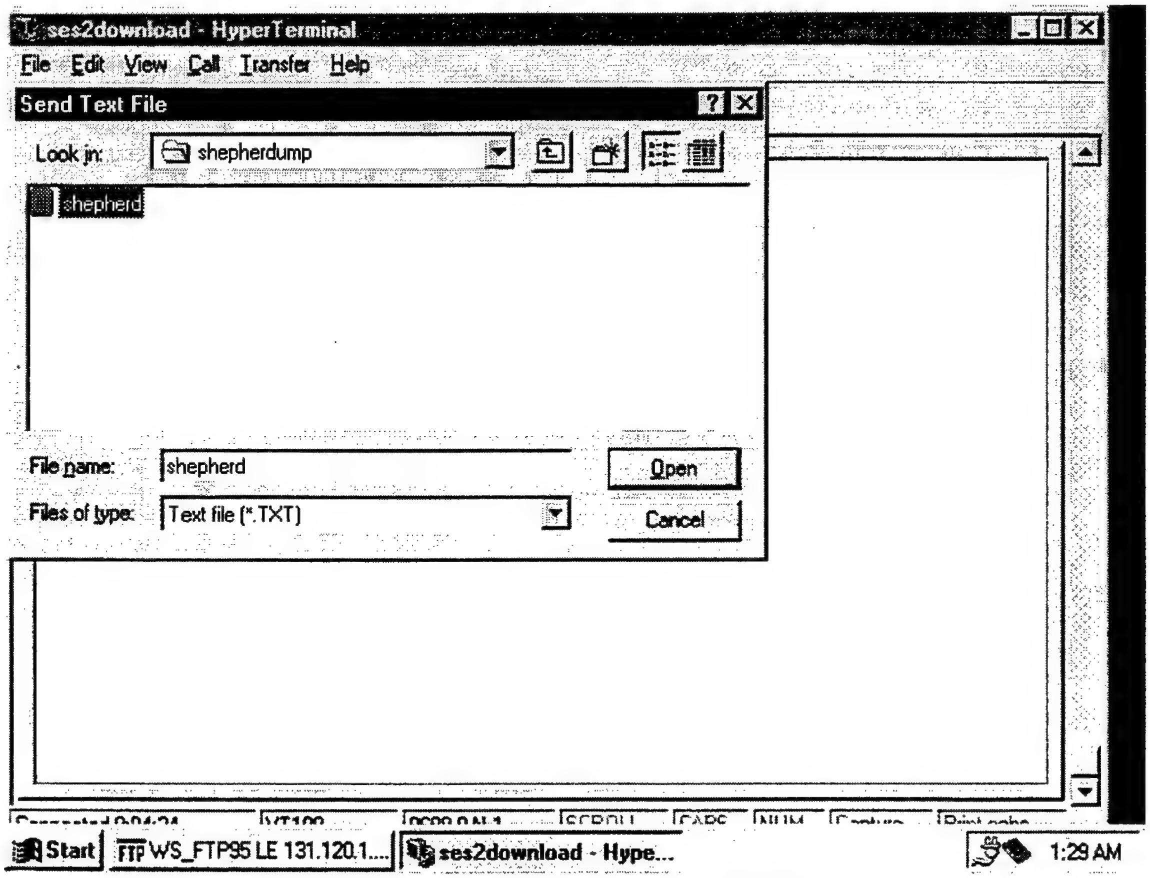

Figure 10: **Send File From shepherdump**

The download process is in motion. The "***red***" transmit light on the RS232 connector to the ***switch box*** will become faint while transmission is in progress. Once the transmission is complete the "***red***" transmit light on the RS232 connector to the ***switch box*** will become a constant red; the ***Hyperterminal window*** will pause during the transmission process. The ***Taurus bug prompt will appear in the HyperTerminal window*** after the transmission is complete.

25. Ensure the ***lever*** on the ***switch box*** is placed on ***console*** (this allows the console to emulate a VT220).

25. Now type *"go"* at the ***Taurus bug prompt*** and ***press the enter key*** (see figure 11). The program that you have previously downloaded will be executed.

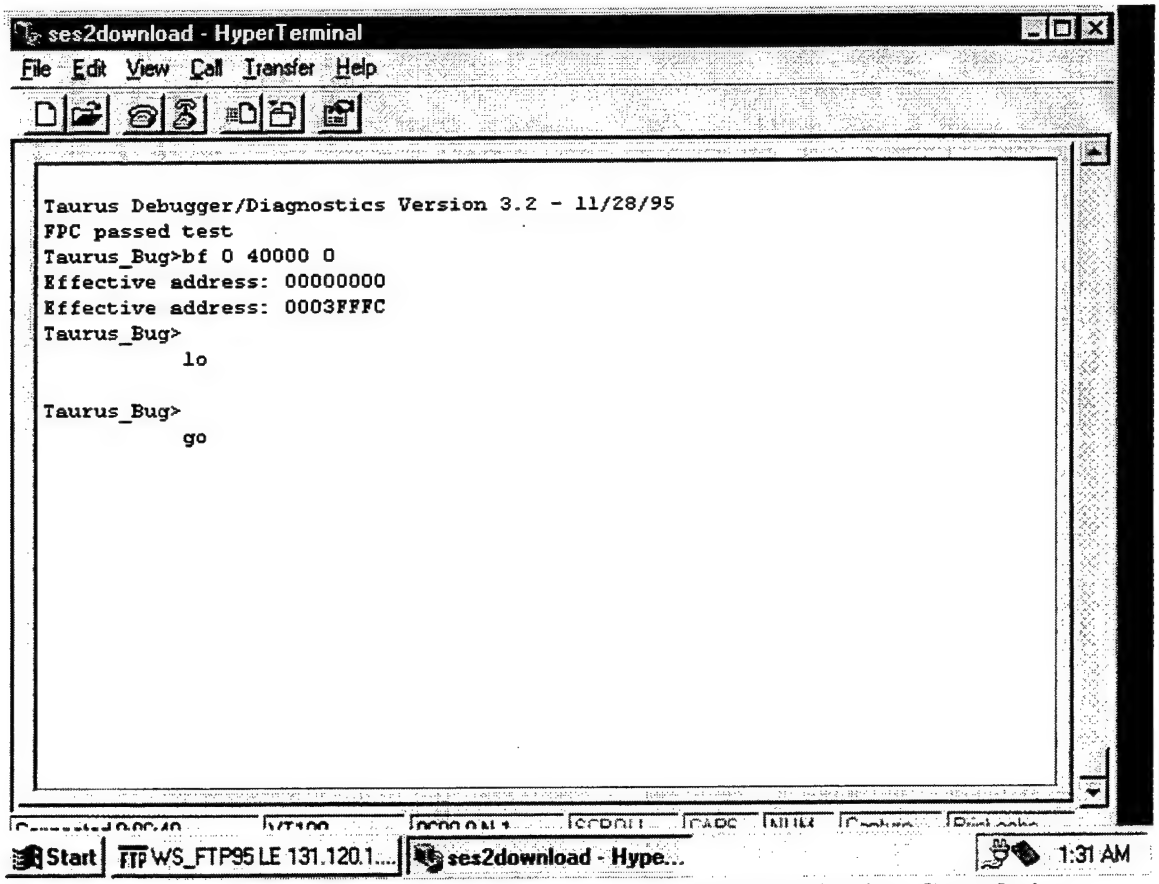

Figure 11: **Taurus Bug Prompt Returns After Transmisssion Completion and the "go" Command is given to Execute the Program.**

After the *"go"* has been given and the execution begins the Shepherd Main Menu appears for your selection.

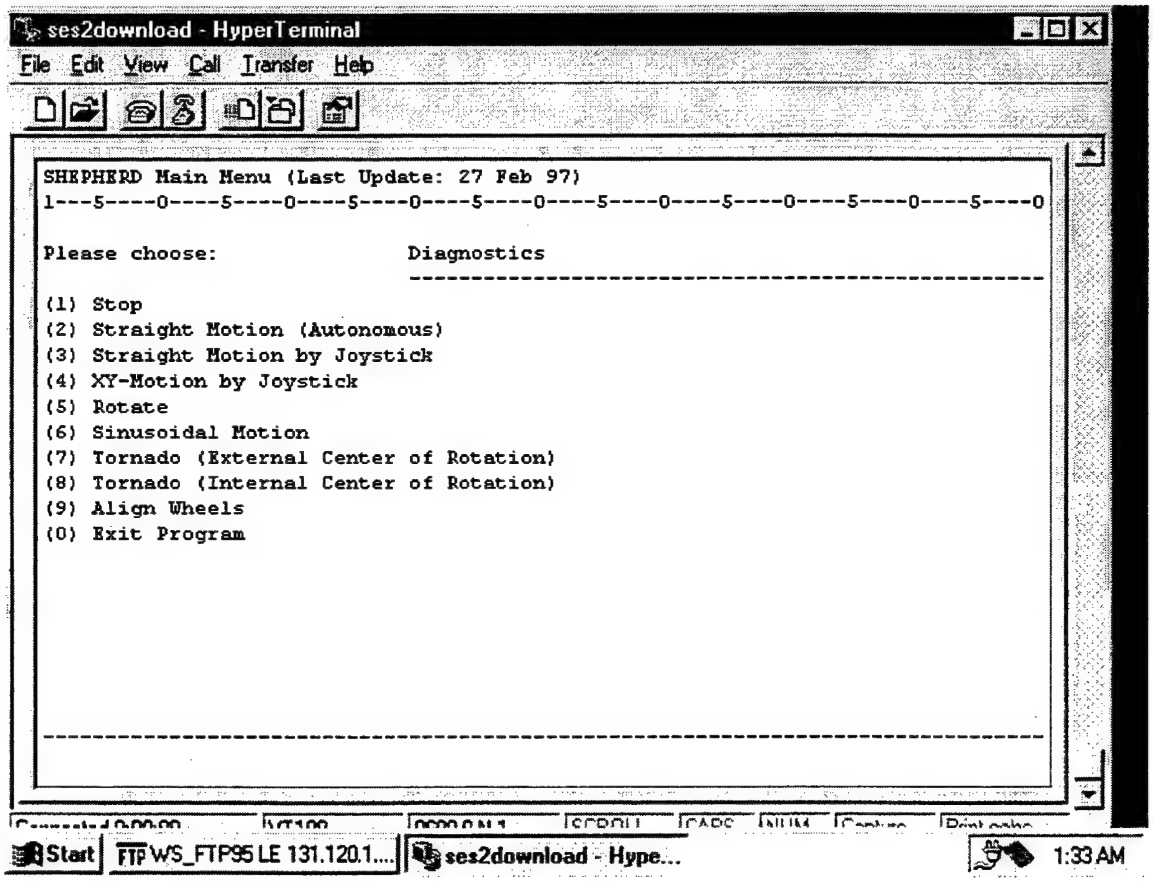

Figure 12: **Shepherd Main Menu**

Note: remember this is a ***quick guide*** and does not provided answers to questions concerning file size, debugger commands, and other requirements or constraints.

APPENDIX L: SENSING SIMULATION DATA

Distance	x	y	theta	psi
0.000	0.400	0.000	0.001	2.355
0.400	0.800	0.001	0.002	2.353
0.800	1.200	0.002	0.005	2.352
1.200	1.600	0.005	0.008	2.350
1.600	2.000	0.008	0.011	2.348
2.000	2.400	0.014	0.015	2.347
2.400	2.800	0.021	0.020	2.345
2.800	3.200	0.029	0.025	2.344
3.200	3.600	0.041	0.031	2.342
3.600	3.999	0.054	0.037	2.341
4.000	4.399	0.070	0.043	2.339
4.400	4.799	0.088	0.050	2.337
4.800	5.198	0.110	0.057	2.336
5.200	5.597	0.134	0.064	2.334
5.600	5.996	0.161	0.072	2.333
6.000	6.395	0.192	0.080	2.331
6.400	6.794	0.225	0.088	2.330
6.800	7.192	0.262	0.096	2.328
7.200	7.590	0.302	0.105	2.326
7.600	7.988	0.346	0.114	2.325
8.000	8.385	0.393	0.123	2.323
8.400	8.782	0.444	0.131	2.322
8.800	9.178	0.498	0.141	2.320
9.200	9.574	0.556	0.150	2.319
9.600	9.969	0.617	0.159	2.317

10.000	10.364	0.682	0.168	2.315
10.400	10.758	0.751	0.177	2.314
10.800	11.151	0.823	0.187	2.312
11.200	11.544	0.899	0.196	2.311
11.600	11.936	0.979	0.205	2.309
12.000	12.327	1.062	0.214	2.308
12.400	12.718	1.149	0.224	2.306
12.800	13.107	1.240	0.233	2.304
13.200	13.496	1.334	0.242	2.303
13.600	13.884	1.431	0.251	2.301
14.000	14.271	1.532	0.260	2.300
14.400	14.657	1.637	0.269	2.298
14.800	15.042	1.745	0.278	2.297
15.200	15.426	1.856	0.286	2.295
15.600	15.810	1.971	0.295	2.293
16.000	16.192	2.089	0.303	2.292
16.400	16.573	2.210	0.312	2.290
16.800	16.953	2.334	0.320	2.289
17.200	17.332	2.461	0.328	2.287
17.600	17.711	2.592	0.336	2.286
18.000	18.088	2.725	0.344	2.284
18.400	18.464	2.862	0.352	2.282
18.800	18.839	3.001	0.359	2.281
19.200	19.213	3.143	0.367	2.279
19.600	19.585	3.288	0.374	2.278
20.000	19.957	3.435	0.381	2.276
20.400	20.328	3.586	0.388	2.275
20.800	20.698	3.738	0.395	2.273
21.200	21.066	3.893	0.402	2.271

21.600	21.434	4.051	0.408	2.270
22.000	21.801	4.211	0.415	2.268
22.400	22.166	4.373	0.421	2.267
22.800	22.531	4.538	0.427	2.265
23.200	22.894	4.704	0.433	2.264
23.600	23.257	4.873	0.438	2.262
24.000	23.619	5.044	0.444	2.260
24.400	23.980	5.217	0.449	2.259
24.800	24.339	5.391	0.454	2.257
25.200	24.698	5.568	0.459	2.256
25.600	25.056	5.746	0.464	2.254
26.000	25.414	5.926	0.469	2.253
26.400	25.770	6.108	0.474	2.251
26.800	26.126	6.291	0.478	2.249
27.200	26.480	6.476	0.482	2.248
27.600	26.834	6.662	0.486	2.246
28.000	27.188	6.849	0.490	2.245
28.400	27.540	7.038	0.494	2.243
28.800	27.892	7.229	0.497	2.242
29.200	28.243	7.420	0.501	2.240
29.600	28.594	7.613	0.504	2.238
30.000	28.944	7.807	0.507	2.237
30.400	29.293	8.001	0.510	2.235
30.800	29.642	8.197	0.513	2.234
31.200	29.990	8.394	0.516	2.232
31.600	30.338	8.592	0.518	2.231
32.000	30.685	8.790	0.521	2.229
32.400	31.032	8.990	0.523	2.227
32.800	31.378	9.190	0.525	2.226

33.200	31.724	9.390	0.527	2.224
33.600	32.070	9.592	0.529	2.223
34.000	32.415	9.794	0.530	2.221
34.400	32.760	9.996	0.532	2.220
34.800	33.105	10.200	0.533	2.218
35.200	33.449	10.403	0.535	2.216
35.600	33.793	10.607	0.536	2.215
36.000	34.137	10.811	0.537	2.213
36.400	34.481	11.016	0.538	2.212
36.800	34.824	11.221	0.538	2.210
37.200	35.167	11.426	0.539	2.209
37.600	35.511	11.632	0.540	2.207
38.000	35.854	11.837	0.540	2.205
38.400	36.197	12.043	0.540	2.204
38.800	36.540	12.249	0.541	2.202
39.200	36.883	12.455	0.541	2.201
39.600	37.225	12.661	0.541	2.199
40.000	37.568	12.867	0.541	2.198
40.400	37.911	13.073	0.541	2.196
40.800	38.254	13.279	0.540	2.194
41.200	38.597	13.484	0.540	2.193
41.600	38.941	13.690	0.539	2.191
42.000	39.284	13.895	0.539	2.190
42.400	39.627	14.100	0.538	2.188
42.800	39.971	14.305	0.537	2.187
43.200	40.315	14.510	0.537	2.185
43.600	40.658	14.714	0.536	2.183
44.000	41.002	14.918	0.535	2.182
44.400	41.347	15.122	0.534	2.180

44.800	41.691	15.325	0.532	2.179
45.200	42.036	15.528	0.531	2.177
45.600	42.381	15.730	0.530	2.176
46.000	42.726	15.932	0.528	2.174
46.400	43.072	16.133	0.527	2.172
46.800	43.418	16.334	0.525	2.171
47.200	43.764	16.535	0.524	2.169
47.600	44.111	16.734	0.522	2.168
48.000	44.457	16.934	0.521	2.166
48.400	44.805	17.132	0.519	2.165
48.800	45.152	17.330	0.517	2.163
49.200	45.500	17.528	0.515	2.161
49.600	45.848	17.724	0.513	2.160
50.000	46.197	17.920	0.511	2.158
50.400	46.546	18.116	0.509	2.157
50.800	46.896	18.310	0.507	2.155
51.200	47.246	18.504	0.505	2.154
51.600	47.596	18.697	0.503	2.152
52.000	47.947	18.889	0.500	2.150
52.400	48.298	19.081	0.498	2.149
52.800	48.649	19.272	0.496	2.147
53.200	49.002	19.461	0.493	2.146
53.600	49.354	19.650	0.491	2.144
54.000	49.707	19.839	0.489	2.143
54.400	50.060	20.026	0.486	2.141
54.800	50.414	20.212	0.484	2.139
55.200	50.769	20.398	0.481	2.138
55.600	51.124	20.582	0.478	2.136
56.000	51.479	20.766	0.476	2.135

56.400	51.835	20.949	0.473	2.133
56.800	52.191	21.131	0.470	2.132
57.200	52.548	21.311	0.468	2.130
57.600	52.905	21.491	0.465	2.128
58.000	53.263	21.670	0.462	2.127
58.400	53.621	21.848	0.460	2.125
58.800	53.980	22.025	0.457	2.124
59.200	54.339	22.201	0.454	2.122
59.600	54.699	22.376	0.451	2.121
60.000	55.059	22.550	0.448	2.119
60.400	55.420	22.723	0.445	2.117
60.800	55.781	22.894	0.442	2.116
61.200	56.143	23.065	0.440	2.114
61.600	56.505	23.235	0.437	2.113
62.000	56.868	23.403	0.434	2.111
62.400	57.231	23.571	0.431	2.110
62.800	57.595	23.738	0.428	2.108
63.200	57.959	23.903	0.425	2.106
63.600	58.323	24.067	0.422	2.105
64.000	58.689	24.231	0.419	2.103
64.400	59.054	24.393	0.416	2.102
64.800	59.420	24.554	0.413	2.100
65.200	59.787	24.714	0.410	2.099
65.600	60.154	24.873	0.407	2.097
66.000	60.522	25.030	0.404	2.095
66.400	60.890	25.187	0.401	2.094
66.800	61.258	25.343	0.398	2.092
67.200	61.627	25.497	0.395	2.091
67.600	61.997	25.650	0.392	2.089

68.000	62.367	25.802	0.389	2.088
68.400	62.737	25.953	0.386	2.086
68.800	63.108	26.103	0.383	2.084
69.200	63.479	26.252	0.380	2.083
69.600	63.851	26.400	0.377	2.081
70.000	64.223	26.546	0.374	2.080
70.400	64.596	26.692	0.371	2.078
70.800	64.969	26.836	0.368	2.077
71.200	65.342	26.979	0.365	2.075
71.600	65.716	27.121	0.362	2.073
72.000	66.091	27.262	0.358	2.072
72.400	66.465	27.402	0.355	2.070
72.800	66.841	27.541	0.352	2.069
73.200	67.216	27.678	0.349	2.067
73.600	67.592	27.815	0.347	2.066
74.000	67.969	27.950	0.344	2.064
74.400	68.346	28.084	0.341	2.062
74.800	68.723	28.217	0.338	2.061
75.200	69.100	28.349	0.335	2.059
75.600	69.478	28.480	0.332	2.058
76.000	69.857	28.610	0.329	2.056
76.400	70.236	28.738	0.326	2.055
76.800	70.615	28.866	0.323	2.053
77.200	70.994	28.992	0.320	2.051
77.600	71.374	29.117	0.317	2.050
78.000	71.754	29.241	0.314	2.048
78.400	72.135	29.365	0.311	2.047
78.800	72.516	29.487	0.308	2.045
79.200	72.897	29.607	0.306	2.044

79.600	73.279	29.727	0.303	2.042
80.000	73.661	29.846	0.300	2.040
80.400	74.043	29.964	0.297	2.039
80.800	74.426	30.080	0.294	2.037
81.200	74.809	30.196	0.292	2.036
81.600	75.192	30.310	0.289	2.034
82.000	75.576	30.424	0.286	2.033
82.400	75.959	30.536	0.283	2.031
82.800	76.344	30.647	0.281	2.029
83.200	76.728	30.757	0.278	2.028
83.600	77.113	30.867	0.275	2.026
84.000	77.498	30.975	0.272	2.025
84.400	77.883	31.082	0.270	2.023
84.800	78.269	31.188	0.267	2.022
85.200	78.655	31.293	0.264	2.020
85.600	79.041	31.397	0.262	2.018
86.000	79.428	31.500	0.259	2.017
86.400	79.815	31.602	0.257	2.015
86.800	80.202	31.703	0.254	2.014
87.200	80.589	31.803	0.251	2.012
87.600	80.976	31.902	0.249	2.011
88.000	81.364	32.000	0.246	2.009
88.400	81.752	32.097	0.244	2.008
88.800	82.141	32.193	0.241	2.006
89.200	82.529	32.289	0.239	2.004
89.600	82.918	32.383	0.236	2.003
90.000	83.307	32.476	0.234	2.001
90.400	83.696	32.568	0.232	2.000
90.800	84.085	32.659	0.229	1.998

91.200	84.475	32.750	0.227	1.997
91.600	84.865	32.839	0.224	1.995
92.000	85.255	32.928	0.222	1.993
92.400	85.645	33.015	0.220	1.992
92.800	86.036	33.102	0.217	1.990
93.200	86.427	33.188	0.215	1.989
93.600	86.817	33.273	0.213	1.987
94.000	87.209	33.357	0.210	1.986
94.400	87.600	33.440	0.208	1.984
94.800	87.991	33.522	0.206	1.982
95.200	88.383	33.603	0.204	1.981
95.600	88.775	33.684	0.201	1.979
96.000	89.167	33.763	0.199	1.978
96.400	89.559	33.842	0.197	1.976
96.800	89.951	33.920	0.195	1.975
97.200	90.344	33.997	0.193	1.973
97.600	90.736	34.073	0.191	1.971
98.000	91.129	34.149	0.189	1.970
98.400	91.522	34.223	0.186	1.968
98.800	91.915	34.297	0.184	1.967
99.200	92.309	34.370	0.182	1.965
99.600	92.702	34.442	0.180	1.964
100.000	93.096	34.513	0.178	1.962
100.400	93.489	34.584	0.176	1.960
100.800	93.883	34.654	0.174	1.959
101.200	94.277	34.723	0.172	1.957
101.600	94.671	34.791	0.170	1.956
102.000	95.066	34.858	0.168	1.954
102.400	95.460	34.925	0.166	1.953

102.800	95.855	34.991	0.165	1.951
103.200	96.249	35.056	0.163	1.949
103.600	96.644	35.120	0.161	1.948
104.000	97.039	35.184	0.159	1.946
104.400	97.434	35.247	0.157	1.945
104.800	97.829	35.309	0.155	1.943
105.200	98.224	35.371	0.154	1.942
105.600	98.620	35.432	0.152	1.940
106.000	99.015	35.492	0.150	1.938
106.400	99.411	35.551	0.148	1.937
106.800	99.806	35.610	0.146	1.935
107.200	100.202	35.668	0.145	1.934
107.600	100.598	35.725	0.143	1.932
108.000	100.994	35.782	0.141	1.931
108.400	101.390	35.838	0.140	1.929
108.800	101.786	35.893	0.138	1.927
109.200	102.182	35.948	0.136	1.926
109.600	102.579	36.002	0.135	1.924
110.000	102.975	36.056	0.133	1.923
110.400	103.372	36.108	0.132	1.921
110.800	103.768	36.161	0.130	1.920
111.200	104.165	36.212	0.128	1.918
111.600	104.562	36.263	0.127	1.916
112.000	104.959	36.313	0.125	1.915
112.400	105.355	36.363	0.124	1.913
112.800	105.752	36.412	0.122	1.912
113.200	106.149	36.461	0.121	1.910
113.600	106.547	36.509	0.119	1.909
114.000	106.944	36.556	0.118	1.907

114.400	107.341	36.603	0.116	1.905
114.800	107.738	36.649	0.115	1.904
115.200	108.136	36.695	0.114	1.902
115.600	108.533	36.740	0.112	1.901
116.000	108.931	36.784	0.111	1.899
116.400	109.328	36.828	0.109	1.898
116.800	109.726	36.872	0.108	1.896
117.200	110.124	36.915	0.107	1.894
117.600	110.521	36.957	0.105	1.893
118.000	110.919	36.999	0.104	1.891
118.400	111.317	37.040	0.103	1.890
118.800	111.715	37.081	0.102	1.888
119.200	112.113	37.121	0.100	1.887
119.600	112.511	37.161	0.099	1.885
120.000	112.909	37.200	0.098	1.883
120.400	113.307	37.239	0.097	1.882
120.800	113.705	37.277	0.095	1.880
121.200	114.103	37.315	0.094	1.879
121.600	114.502	37.353	0.093	1.877
122.000	114.900	37.389	0.092	1.876
122.400	115.298	37.426	0.091	1.874
122.800	115.697	37.462	0.089	1.872
123.200	116.095	37.497	0.088	1.871
123.600	116.494	37.532	0.087	1.869
124.000	116.892	37.567	0.086	1.868
124.400	117.291	37.601	0.085	1.866
124.800	117.689	37.635	0.084	1.865
125.200	118.088	37.668	0.083	1.863
125.600	118.486	37.701	0.082	1.861

APPENDIX M: SENSING SIMULATION CODE (MAIN2.CC)

```
// File: main2.cc

// Name: Edward Mays

// Sensing Simulation

// Unix

// GCC

// Date: 26 August 1997

//

// Description

// THIS PROGM SIMULATES THE MOVEMENT OF A SQARE OBJECT ALONG A
//PATH. THE OBJECT'S PATH DIRECTION (THETA) IS CHANGING, AS IS THE
//OBJECTS ORIENTATION (PSI). LINE TRACKING IS USED AND THE X-AXIS IS
THE //REFERENCE LINE. THE REFERENCE LINE IS INCREMENTED BY 40
UNITS IN THE

// ----------------------------------

// Header file info

// ----------------------------------

#include <iostream.h>

#include <math.h>

#include <fstream.h>

#include <stdio.h>



#define PI    3.14159265358979323846

#define RAD   57.29577951308232087684
```

```
double deltaTime = 0.01;// 0.01
double Vel = 40.0;
double omega = -0.1570796327;

FILE *f0, *f1, *f2, *f3, *f4, *f5,*f6 ;      //PTR TO FILE FOR OUTPUT DATA



//structure to hold configuration including x, y, theta, and kappa


typedef struct{
   double x;
   double y;   }
  POINT;


  typedef struct{
   POINT Point;
  double Theta;
  double Kappa;
   double Psi;
   }
  CONFIGURATION;



  // -----------------------------------
  //Function: GetSmooth
  //Return Value:n/a
```

```
//Parameters:  function parm  list
//Purpose: gets users input for s0/smoothness
// ----------------------------------
double GetSmooth(double &s0)
{



cout << "enter your value for smoothness (negatives not allowed)" <<endl;
 cin >> s0;
 return (s0 >= 0.0);

}// GetSmooth

// ----------------------------------
//Function: InitConfig
//Return Value:n/a
//Parameters:  function parm  list
//Purpose:  SETS INITIAL CONFIGURATION
// ----------------------------------
void    InitConfig(CONFIGURATION&    q_init,    CONFIGURATION&
q_xaxis,CONFIGURATION& qbody,CONFIGURATION& qfrontR,

    CONFIGURATION& qfrontL, CONFIGURATION& qrearR, CONFIGURATION&
qrearL, CONFIGURATION& qsnapshot,

        double &s0, double &deltaS)
{
cout<<"Setting the initial configuration"
```

```
<<"x=0, y = 0, theta = 0, and kappa = 0 "<<endl;

q_init.Point.x = 0.0;
q_init.Point.y = 0.0;
q_init.Theta  = 0.0;
q_init.Kappa  = 0.0;
q_init.Psi  = 2.356219449; /* 3*PI/4.0 */

cout<<"Setting the reference line configuration"
<<"x = 0, y = 40, theta = 0 , and kappa = 0 "<<endl;

q_xaxis.Point.x  = 0.0;
q_xaxis.Point.y  = 40.0;
q_xaxis.Theta  = 0.0;
q_xaxis.Kappa  = 0.0;

//individual wheels
qfrontR.Point.x = 40;  /* wheel1 */
qfrontR.Point.y = -40;

qfrontL.Point.x = 40;  /* wheel2 */
qfrontL.Point.y = 40;

qrearR.Point.x = -40;  /* wheel3 */
qrearR.Point.y = -40;
```

```
qrearL.Point.x = -40;  /* wheel 4 */
qrearL.Point.y =  40;

qsnapshot.Point.x = 0.0;
qsnapshot.Point.y = 0.0;

cout<<"Enter size constant for smoothness <return>"<<endl;
GetSmooth(s0);

cout<<"Entering Step size constant deltaS(deltaS=Vel*deltaT)."<<endl;
deltaS = Vel*deltaTime;//.05 orig


}// InitConfig



// ----------------------------------
//Function: CreateConst
//Return Value:n/a
//Parameters:  function parm  list
//Purpose:  create constants for
//          steering function dk/ds
// ----------------------------------
void CreateConst(double &a, double &b, double &c, double &s0)
```

```
{
double k;
 k = 1.0/s0;  //all consts by def, including curvature
 a = 3.0*k;
 b = 3.0*k*k;
c = k*k*k;
}// CreateConst
```

```
// ----------------------------------
//Function: GetSteerL
//Return Value:n/a
//Parameters:  function parm  list
//Purpose:     lambda=dk/ds  (LINEAR STEERING FUNCTION)
// ----------------------------------

double GetSteerL(double &a, double &b, double &c, CONFIGURATION& q,
    CONFIGURATION& q_xaxis)
{
```

```
double delta_r;

delta_r = -(q.Point.x - q_xaxis.Point.x)*sin(q_xaxis.Theta) +
        (q.Point.y - q_xaxis.Point.y)*cos(q_xaxis.Theta);

return (-(a*q.Kappa + b*(q.Theta -  q_xaxis.Theta) + c*delta_r));
} //GetSteerL




// ----------------------------------
//Function: GetDeltakappa
//Return Value:n/a
//Parameters:  function parm  list
//Purpose:   DETERMINES THE KAPPA DIFFERENCE PER INCREMENT OF S
// ----------------------------------

double GetDeltakappa(double &Dk_Ds, double &deltaS, double &deltakappa)
{
deltakappa = Dk_Ds*deltaS;
return(deltakappa);

}//GetDeltakappa
```

```
// ----------------------------------
//Function: returnkappa
//Return Value:n/a
//Parameters:  function parm  list
//Purpose:  CALCULATES NEW VALUE FOR KAPPA USING deltaK
// ----------------------------------

CONFIGURATION returnkappa(double &deltakappa, CONFIGURATION &q)
{
 q.Kappa = q.Kappa + deltakappa;
 return q;

}//returnkappa




// ----------------------------------
//Function: GetS
//Return Value:n/a
//Parameters:  function parm  list
//Purpose:  INCREMENTS S THROUGH EACH ITERATION OF THE WHILE LOOP
// ----------------------------------

double GetS(double &s, double &deltaS)
{
```

```
 s = s + deltaS;
return s;

}//GetS

// ----------------------------------
//Function: GetDeltaTheta
//Return Value:n/a
//Parameters:  function parm  list
//Purpose: COMPUTES CHANGE IN THETA PER INCREMENT OF S
// ----------------------------------

double GetDeltaTheta(CONFIGURATION &q, double &deltaS, double &deltaT)
{
deltaT =  q.Kappa*deltaS;
return(deltaT);

}//GetDeltaTheta

// ----------------------------------
//Function: Circ
//Return Value:n/a
//Parameters:  function parm  list
//Purpose:  Circ function from notes 6.29
// ----------------------------------
```

```
void Circ(double Length, double alpha, CONFIGURATION &q)
{
double alpha2, alpha4;
 alpha2=alpha*alpha;
alpha4=alpha2*alpha2;



//configuration q1
q.Point.x = (1.0 - alpha2/6.0 + alpha4/120.0)*Length;
q.Point.y = (0.5 - alpha2/24.0 + alpha4/720.0)*Length*alpha;
q.Theta  = alpha;


}//Circ


// -----------------------------------
//Function: Compose
//Return Value:n/a
//Parameters:  function parm  list
//Purpose: updates the configuration and computes new config (notes 6.2)
// -----------------------------------
CONFIGURATION Compose(CONFIGURATION& q1,CONFIGURATION& q2,CONFIGURATION& q3, double& s,double& deltaTime)
```

```
{   double x,y,
sinTheta = sin(q1.Theta),
cosTheta = cos(q1.Theta);

x = q1.Point.x + q2.Point.x*cosTheta - q2.Point.y*sinTheta;
y = q1.Point.y + q2.Point.x*sinTheta + q2.Point.y*cosTheta;
q3.Point.x = x;
q3.Point.y = y;
q3.Theta = q1.Theta + q2.Theta;

 q3.Psi = q1.Psi + (omega * deltaTime);   /* how to handle move left/right? */

 fprintf(f6,"%10.3f %10.3f %10.3f %10.3f %10.3f\n",
         s,q3.Point.x, q3.Point.y,q3.Theta, q3.Psi);
return q3;

}// end Compose



CONFIGURATION   Compose2(CONFIGURATION&   q1,CONFIGURATION&   q2,
CONFIGURATION& q3) /*position */
{   double x,y,
 sinTheta = sin(q1.Psi),
cosTheta = cos(q1.Psi);
```

```
x = q1.Point.x + q2.Point.x*cosTheta - q2.Point.y*sinTheta;
y = q1.Point.y + q2.Point.x*sinTheta + q2.Point.y*cosTheta;

q3.Point.x = x;
q3.Point.y = y;
return q3;
}// end Compose2



// -----------------------------------
//Function: Openfile
//Return Value:n/a
//Parameters:  function parm  list
//Purpose: To compute transposition
// -----------------------------------
void Openfile()
{
f0 = fopen("drk.dat","w");
f1 = fopen("wheel1.dat","w");
f2 = fopen("wheel2.dat","w");
f3 = fopen("wheel3.dat","w");
f4 = fopen("wheel4.dat","w");
f5 = fopen("composite.dat","w");
f6 = fopen("psi.dat","w");
}// Openfile
```

```
// ----------------------------------
//Function: Print
//Return Value:n/a
//Parameters:  function parm  list
//Purpose: To compute transposition
// ----------------------------------
void printFile(FILE *f,CONFIGURATION &q)
{

 fprintf(f,"%10.3f %10.3f\n",
        q.Point.x, q.Point.y);


}// printFile



// ----------------------------------
//Function: blankLine
//Return Value:n/a
//Parameters:  function parm  list
//Purpose: To compute transposition
// ----------------------------------
void blankLine(FILE *f)
{
```

```
fprintf(f,"\n");

}// blankLine

// ----------------------------------

//Function: updateWheels
//Return Value:n/a
//Parameters:  function parm  list
/Purpose:  create constants for
//        steering function dk/ds
// ----------------------------------
void updateWheels(CONFIGURATION& qbody, CONFIGURATION& qfrontR,
CONFIGURATION& qfrontL, CONFIGURATION qrearR, CONFIGURATION qrearL,
CONFIGURATION& qwheel1, CONFIGURATION& qwheel2, CONFIGURATION&
qwheel3, CONFIGURATION& qwheel4, CONFIGURATION& q3,int& s2)
{    printFile(f0,qbody);


qwheel1 = Compose2(qbody,qfrontR,q3);
printFile(f1,qwheel1);
blankLine(f1);
qwheel2 = Compose2(qbody,qfrontL,q3);
printFile(f2,qwheel2);
```

```
blankLine(f2);
qwheel3 = Compose2(qbody, qrearR, q3);
printFile(f3,qwheel3);
blankLine(f3);
qwheel4 = Compose2(qbody, qrearL,q3);
printFile(f4,qwheel4);
blankLine(f4);

if((s2==0.0)||(s2%100==0)){

printFile(f5,qwheel1);
printFile(f5,qwheel2);
printFile(f5,qwheel4);
printFile(f5,qwheel3);
printFile(f5,qwheel1);
blankLine(f5);
}
}// updateWheels

int main()
```

```
{ CONFIGURATION q, q_xaxis, New_q, qbody, qfrontR, qfrontL, qrearR, qrearL,
qwheel1, qwheel2, qwheel3, qwheel4, qsnapshot,q3;


int ix,s2,counter;

double a, b, c;                     //constants equation 6.3

double s,s0;                        //s0 is smoothness, s is the incremental step

double Dk_Ds;

double deltaS;

double deltaT;

double deltaK;

const double Sdig = 0.001;              //const used for prec/toler

double smax=400.0;


InitConfig(q, q_xaxis,  qbody, qfrontR, qfrontL, qrearR, qrearL,qsnapshot,s0, deltaS);
//configuaration set up


Openfile();

printFile(f0,q);

printFile(f1,qfrontR);                      //write initial config to file

printFile(f2,qfrontL);

printFile(f3,qrearR);

 printFile(f4,qrearL);



//printFile(f5,qfrontR);

//printFile(f5,qfrontL);
```

```
//printFile(f5,qrearL);
//printFile(f5,qrearR);

printFile(f5,qfrontR);
blankLine(f5);

CreateConst(a, b, c, s0);                //calcs consts
s = 0.0;
s2=s;
counter=0;
for(ix=0; ix<10;ix++){
do
{//

Dk_Ds = GetSteerL(a, b, c, q, q_xaxis);        //calculates lambda =dk/ds
deltaK = GetDeltakappa(Dk_Ds, deltaS, deltaK);  //lambda*deltaS
returnkappa(deltaK, q);                  //Kappa <= kappa + deltaK
deltaT = GetDeltaTheta(q, deltaS, deltaT);      //Theta <= Theta + deltaT
Circ(deltaS, deltaT, New_q);             //cir
qbody = Compose(q, New_q, q, s, deltaTime);          //compose
updateWheels(qbody,qfrontR, qfrontL, qrearR,
          qrearL,qwheel1, qwheel2, qwheel3,
          qwheel4,q3,s2);
GetS(s, deltaS);
s2=s;
```

```
}while (s<smax);
// }while ((fabs(q.Point.y) > Sdig)||(fabs(q.Theta) > Sdig) ||
//       (fabs(q.Kappa) > Sdig));
s=0.0;
s2=s;
q_xaxis.Point.y  = q_xaxis.Point.y + 40.0;


if(ix%2==0){
q_xaxis.Theta   = PI;
q.Theta   = PI;
omega = fabs(omega);
}else{
q_xaxis.Theta   = 0.0;
q.Theta   = 0.0;
omega = -omega;
}
}
fclose(f0);
fclose(f1);
fclose(f2);
fclose(f3);
fclose(f4);
fclose(f5);
fclose(f6);
return 0;
}//end main2.cc
```

APPENDIX N: INPUT VS. OUPUT VELOCITY

-1024 -87.657

-1023 -87.429

-1020 -86.975

-1015 -86.747

-1010 -86.406

-1005 -85.838

-1000 -85.383

-900 -76.856

-800 -67.998

-700 -59.082

-600 -51.048

-500 -42.748

-400 -34.107

-375 -31.834

-350 -29.673

-300 -25.580

-250 -21.260

-225 -19.100

-200 -17.053

-175 -14.780

-150 -12.506

-125 -10.459

-100 -8.413

-90 -7.503

-80 -6.707

-70 -5.798

-60 -5.002

-50 -4.092

-40 -3.297

-30 -2.387

-20 -1.591

-10 -.682

-5 -.341

-1 -.113

0.0 0.0

1023 87.429

1020 86.975

1015 86.747

1010 86.406

1005 85.838

1000 85.383

900	76.856
800	67.998
700	59.082
600	51.048
500	42.748
400	34.107
375	31.834
350	29.673
300	25.580
250	21.260
225	19.100
200	17.053
175	14.780
150	12.506
125	10.459
100	8.413
90	7.503
80	6.707
70	5.798
60	5.002
50	4.092
40	3.297

30 2.387

20 1.591

10 .682

5 .341

1 .113

APPENDIX O: INPUT VS OUTPUT STEERING RATES

A. DESIRED INPUT RATE VS ACTUAL (BOTH ESTIMATED AND SOFTWARE DEPENDENT)

Desired rate of turn	Time Stop watch (sec)	Estimated Rate (rad/s)	Software Measured Rate (rad/s, average)
1	6.0	1.00000	0.98174
2	3.5	1.79485	1.95667
3	2.19	2.86849	2.93160
5	1.69	2.71716	3.90653
5.5	No data	No data	4.88828
10	No data	No data	5.23598
20	No data	No data	5.23598
30	No data	No data	5.23598

Figure A.1 Inputs and results from massaged data (error). No data entries exist because the revolutions were too fast for hand timing.

B. DESIRED INPUT RATE VS OUPUT FOR EACH WHEEL (SOFTWARE DEPENDENT

Below M5, M6, M7, and M8 reperesent the steering motors for wheel 1, wheel 2, wheel3, and wheel4 respectively.

Desired Rate of turn	M5 Rate	M6 Rate	M7 Rate	M8 Rate
1	1.002	.9975	.9965	.999
2	2.0025	1.997	1.990	1.997
3	3.005	2.995	2.9975	3.005
4	4.004	3.996	3.9925	3.996
5	5.008	5.002	4.9985	5.002

Desired Rate of turn	M5 Rate	M6 Rate	M7 Rate	M8 Rate
5.1	5.1035	5.0935	5.093	5.0955
5.2	5.2065	5.198	5.1955	5.198
5.3	5.238	5.235	5.234	5.235
5.4	5.238	5.235	5.234	5.235
-1	-1.001	-1.002	-1.002	-1.002
-2	-2.001	-2.003	-2.003	-2.004
-3	-3.0055	-3.006	-3.0055	-3.006
-4	-4.007	-4.006	-4.0015	-4.008
-5	-5.010	-5.010	-5.009	-5.010
-5.1	-5.107	-5.110	-5.1035	-5.112
-5.2	-5.2105	-5.2125	-5.2075	-5.214
-5.3	-5.238	-5.2415	-5.237	-5.2535
-5.4	-5.238	-5.2415	-5.237	-5.2535

Figure A.2: Desired (commanded) rate of turn vs. actual "free floating" motor rate.

C. DESIRED INPUT (DIGIT MANIPULATION) VS OUTPUT RATE

Input (digits)	M5 Rate (rad/s)	M6 Rate (rad/s)	M7 Rate (rad/s)	M8 Rate (rad/s)
10	.056	.045	.044	.045
20	.102	.101	.100	.100
30	.152	.143	.143	.143
40	.205	.203	.203	.203
50	.261	.250	.248	.249
60	.306	.306	.305	.305
70	.363	.350	.349	.350
80	.409	.408	.407	.407
90	.466	.454	.452	.455
100	.511	.510	.510	.510
200	1.024	1.022	1.022	1.022

Input (digits)	M5 Rate (rad/s)	M6 Rate (rad/s)	M7 Rate (rad/s)	M8 Rate (rad/s)
300	1.539	1.533	1.533	1.533
400	2.049	2.046	2.045	2.045
500	2.561	2.556	2.556	2.556
600	3.074	3.068	3.067	3.068
700	3.584	3.579	3.579	3.579
800	4.097	4.092	4.091	4.092
900	4.610	4.602	4.602	4.602
1000	5.124	5.116	5.114	5.116
1010	5.174	5.172	5.170	5.172
1020	5.226	5.218	5.216	5.218
1021	5.233	5.226	5.222	5.225
1022	5.235	5.233	5.231	5.232
1023	5.237	5.235	5.234	5.235

Figure A.3: Desired (commanded) rate of turn vs. actual "free floating" motor rate for each wheel using input digits .

D. WHEEL 4 ROTATION DATA

	Clockwise Rotation	Counterclockwise Rotation
1	000.867	360.390
2	000.878	360.390
3	000.976	360.363
4	000.933	360.414
5	000.984	360.371
6	000.992	360.394
7	000.992	360.453
8	000.902	360.394
9	000.996	360.445
10	000.996	360.476
11	001.003	360.402

	Clockwise Rotation	Counterclockwise Rotation
12	000.917	360.417
13	000.996	360.433
14	001.003	360.468
15	000.972	360.468
16	000.863	360.472
17	000.968	360.398
18	000.941	360.425
19	000.957	360.480
20	000.937	360.480

Figure A.4: Wheel 4 data based on position of rest after direction of turn.

LIST OF REFERENCES

1. Review & Outlook, *The Land Mine Dilemma,* The Wall Street Journal, May 7, 1996, Page A18.

2. Tom Nasland and John Barry, *Buried Terror*, Newsweek Special Report, Newsweek, April 8, 1996, pp. 24-27.

3. Steven Ashley, *Searching for Land Mines*, Mechanical Engineering, April 1996, pp. 62-67.

4. X.K. Maruyama, *Technologies in Support of International Peace Operations – Military Technologies for Ground Forces,* Naval Postgraduate School, February 1996.

5. Proceedings of Autonomous Vehicles in Mine Countermeasures Symposium, Naval Postgraduate School, April 4-7, 1995.

6. Evaluation of the Individual Demonstrator performance at the Unexploded Ordnance Advanced Technology Demonstration Program at Jefferson Proving Ground (Phase I), Naval Explosive Ordnance Disposal Technology Division, 1995.

7. Kanayama, Y., *Introduction to Motion Planning lecture Notes of CS4313*, Department of Computer Science, Naval Postgraduate School, 1996.

8. Mitsubishi Heavy Industries, Takasago Research Labs, Rotary Vehicle: Wheel Design Structure Memo, December 1995, paragraph 3.2.

9. Mitsubishi Heavy Industries, Takasago Research Labs, Rotary Vehicle: Wheel Design Structure Memo, December 1995, Annex.

10. Stallings, W., *Computer Organization and Architecture*, Third Edition, Macmillan Publishing, New York, 1993, pp. 90-112.

11. Taurus 68040/68060 VMEbus Single Board Computer, User's Manual, Omnibyte Corporation, Chicago, Illinois, March 1995, pp. 2-1, 2-47, and 2-53.

12. Mays, E., *Shepherd Chronology and Update for CS4920: Seminar in Military Robotics*, Department of Computer Science, Naval Postgraduate School, May 1997.

13. Acromag Series 9210/9215, Analog Output Board, User's Manual, Acromag Incorporated, Wixom Michigan, January 1994, pp. 1.1-1.2.

14. Acromag Series 9421, VME Isolated Digital Input Board, User's Manual, Acromag Incorporated, Wixom Michigan, January 1994, pp. 1.1-1.2.

15. VMIVME-2170A, 32 Bit Optically Coupled Digital Output Board, Instructional Manual, VME Microsystems International Corporation, Huntsville Alabama, February 1994, page 4-2.

16. User Manual, IP-Quadrature, Four Channel Quadrature Decoder Industry Pack, Green Spring Computers, Menlo Park California, 1995, pp. 7-10 and 20-27.

17. Elgerd, O., I., Control Systems Theory, McGraw Hill, New York, 1967, pp. 183-186.

18. Ibid., p. 185.

19. Ibid., pp. 60 and 128.

20. OMNIBug (020Bug) Monitor/Debugger program, Omnibyte Corporation, Chicago, Illinois, March 1995, pp. 1-4, A1, and A3.

21. Taurus 68040/68060 VMEbus Single Board Computer, User's Manual, Omnibyte Corporation, Chicago, Illinois, March 1995, pp. 2-13 through 2-18.

22. Ibid., p. 2-16.

23. Kanayama, Y., *"Rotary Vehicle" That Moves with Three Degrees of Freedom*, Department of Computer Science, Naval Postgraduate School, 1996.

24. Kanayama, Y. and Yun, X., Rigid Body Motion Analysis towards Rotary Vehicle, Naval Postgraduate School, 1997.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center..2
   8725 John J. Kingman Road, Ste 0944
   Ft. Belvoir, VA 22060-6218

2. Dudley Knox Library 2
   Naval Postgraduate School
   411 Dyer Rd.
   Monterey, CA 93943-5101

3. Chairman, Code CS 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

4. Dr. Xiaoping Yun, Code EC/YX 1
   Electrical Engineering Department
   Naval Postgraduate School
   Monterey, CA 93943

5. Dr. Robert B. McGhee, Code CS/Mz 1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

6. Dr. Yutaka Kanayama, Code CS/Kz..1
   Computer Science Department
   Naval Postgraduate School
   Monterey, CA 93943

7. Major Edward J. Mays 4
PM TRCS
Bldg 455 (Oceanport Ave)
ATTN: SFAE-C3S-TRC-EPLRS (Major Mays)
Fort Monmouth, N.J. 07703-5504
Inbound - 15 October 97

8. Lieutenant Ferdinand A. Reid.. 4
Department Head Class 150
SWOSCOLCOM
446 Cushing Road
Newport, RI 02841-1209

9. Director, Training and Education 1
MCCDEC, Code C46
1019 Elliot Road
Quantico, VA 22134-5027

10. Director, Marine Corps Research Center 2
MCCDEC, Code C40RC
2040 Broadway Street
Quantico, VA 22134-5107

11. Director Studies and Analysis Division.. 1
MCCDC, Code C45
3300 Russell Road
Quantico, VA 22134-5130

12. Marine Corps Representative.. 1
Naval Postgraduate School
Code 037, Bldg. 234, HA-220
699 Dyer Road
Monterey, CA 93940

13. Marine Corps Tactical Systems Support Activity.. 1
Technical Advisory Branch
Attn: Maj J. C. Cummiskey
Box 555171
Camp Pendleton, CA 92055-5080